```
# import keras
# from keras.datasets import cifar10
# from keras.models import Model, Sequential
# from keras.layers import Dense, Dropout, Flatten, Input, AveragePooling2D, merge, Activa
# from keras.layers import Conv2D, MaxPooling2D, BatchNormalization
# from keras.layers import Concatenate
# from keras.optimizers import Adam

# 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.image import load_img
from keras.preprocessing.image import img_to_array
from keras.preprocessing.image import ImageDataGenerator
from keras import regularizers
from matplotlib import pyplot


# this part will prevent tensorflow to allocate all the avaliable GPU Memory
# backend
import tensorflow as tf


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


# Load CIFAR10 Data
(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
170500096/170498071 [==============================] - 6s 0us/step
170508288/170498071 [==============================] - 6s 0us/step
```

```
X_train.shape
```

```
(50000, 32, 32, 3)
```

```
X_test.shape
```

```
(10000, 32, 32, 3)
```

```
# Dense Block
def denseblock(input, num_filter = 12, dropout_rate = 0.2):
    global compression
    temp = input
    for _ in range(l):
        BatchNorm = layers.BatchNormalization()(temp)
        relu = layers.Activation('relu')(BatchNorm)
        Conv2D_3_3 = layers.Conv2D(int(num_filter*compression), (3,3), use_bias=False ,pad
        if dropout_rate>0:
            Conv2D_3_3 = layers.Dropout(dropout_rate)(Conv2D_3_3)
        concat = layers.Concatenate(axis=-1)([temp,Conv2D_3_3])

        temp = concat

    return temp

## transition Blosck
def transition(input, num_filter = 12, dropout_rate = 0.2):
    global compression
    BatchNorm = layers.BatchNormalization()(input)
    relu = layers.Activation('relu')(BatchNorm)
    Conv2D_BottleNeck = layers.Conv2D(int(num_filter*compression), (1,1), use_bias=False ,
    if dropout_rate>0:
         Conv2D_BottleNeck = layers.Dropout(dropout_rate)(Conv2D_BottleNeck)
    avg = layers.AveragePooling2D(pool_size=(2,2))(Conv2D_BottleNeck)
    return avg

#output layer
def output_layer(input):
    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 = 0.2
l = 12
input = layers.Input(shape=(img_height, img_width, channel,))
First_Conv2D = layers.Conv2D(num_filter, (3,3), use_bias=False ,padding='same')(input)

First_Block = denseblock(First_Conv2D, num_filter, dropout_rate)
First_Transition = transition(First_Block, num_filter, dropout_rate)

Second_Block = denseblock(First_Transition, num_filter, dropout_rate)
Second_Transition = transition(Second_Block, num_filter, dropout_rate)

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

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


#https://arxiv.org/pdf/1608.06993.pdf
from IPython.display import IFrame, YouTubeVideo
YouTubeVideo(id='-W6y8xnd--U', width=600)
```


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

```
      conv2d_48 (Conv2D)             (None, 4, 4, 6)      2916        ['activation_47[0

      dropout_47 (Dropout)           (None, 4, 4, 6)      0           ['conv2d_48[0][0]

      concatenate_44 (Concatenate)   (None, 4, 4, 60)     0           ['concatenate_43[(
                                                                       'dropout_47[0][0

      batch_normalization_48 (BatchN  (None, 4, 4, 60)    240         ['concatenate_44[(
      ormalization)

      activation_48 (Activation)     (None, 4, 4, 60)     0           ['batch_normaliza

      conv2d_49 (Conv2D)             (None, 4, 4, 6)      3240        ['activation_48[0

      dropout_48 (Dropout)           (None, 4, 4, 6)      0           ['conv2d_49[0][0]

      concatenate_45 (Concatenate)   (None, 4, 4, 66)     0           ['concatenate_44[(
                                                                       'dropout_48[0][0

      batch_normalization_49 (BatchN  (None, 4, 4, 66)    264         ['concatenate_45[(
      ormalization)

      activation_49 (Activation)     (None, 4, 4, 66)     0           ['batch_normaliza

      conv2d_50 (Conv2D)             (None, 4, 4, 6)      3564        ['activation_49[0

      dropout_49 (Dropout)           (None, 4, 4, 6)      0           ['conv2d_50[0][0]
```

```
concatenate_46 (Concatenate)   (None, 4, 4, 72)     0           ['concatenate_45[0
                                                                 'dropout_49[0][0

batch_normalization_50 (BatchN  (None, 4, 4, 72)    288         ['concatenate_46[0
ormalization)

activation_50 (Activation)     (None, 4, 4, 72)     0           ['batch_normaliza

conv2d_51 (Conv2D)             (None, 4, 4, 6)      3888        ['activation_50[0

dropout_50 (Dropout)           (None, 4, 4, 6)      0           ['conv2d_51[0][0]

concatenate_47 (Concatenate)   (None, 4, 4, 78)     0           ['concatenate_46[0
                                                                 'dropout_50[0][0

batch_normalization_51 (BatchN  (None, 4, 4, 78)    312         ['concatenate_47[0
ormalization)

activation_51 (Activation)     (None, 4, 4, 78)     0           ['batch_normaliza

average_pooling2d_3 (AveragePo  (None, 2, 2, 78)    0           ['activation_51[0
oling2D)

flatten (Flatten)              (None, 312)          0           ['average_pooling

dense (Dense)                  (None, 10)           3130        ['flatten[0][0]']

==================================================================================
Total params: 118,918
```

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


model.fit(X_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(X_test, y_test))
```

```
Epoch 1/10
391/391 [==============================] - 25s 65ms/step - loss: 1.7582 - accuracy: 0
Epoch 2/10
391/391 [==============================] - 24s 61ms/step - loss: 1.4103 - accuracy: 0
Epoch 3/10
391/391 [==============================] - 24s 61ms/step - loss: 1.2338 - accuracy: 0
Epoch 4/10
391/391 [==============================] - 24s 61ms/step - loss: 1.1185 - accuracy: 0
Epoch 5/10
391/391 [==============================] - 24s 61ms/step - loss: 1.0438 - accuracy: 0
Epoch 6/10
391/391 [==============================] - 24s 61ms/step - loss: 0.9919 - accuracy: 0
Epoch 7/10
391/391 [==============================] - 24s 61ms/step - loss: 0.9513 - accuracy: 0
Epoch 8/10
391/391 [==============================] - 24s 61ms/step - loss: 0.9180 - accuracy: 0
```

```
Epoch 9/10
391/391 [==============================] - 24s 61ms/step - loss: 0.8895 - accuracy: 0
Epoch 10/10
391/391 [==============================] - 24s 61ms/step - loss: 0.8649 - accuracy: 0
<tensorflow.python.keras.callbacks.History at 0x7efc85359240>
```

```
# Test the model
score = model.evaluate(X_test, y_test, verbose=1)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
313/313 [==============================] - 2s 7ms/step - loss: 1.7172 - accuracy: 0.5
Test loss: 1.7172198295593262
Test accuracy: 0.5551999807357788
```

```
# Save the trained weights in to .h5 format
model.save_weights("DNST_model.h5")
print("Saved model to disk")
```

```
Saved model to disk
```

```
# free model variable
del model
```

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 Dense Layers (also called fully connected layers), or DropOut.
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.

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


# Load CIFAR10 Data
(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]

# Apply one hot encoding
y_train = tf.keras.utils.to_categorical(y_train, num_classes)
y_test = tf.keras.utils.to_categorical(y_test, num_classes)


X_train.shape
```

```
     (50000, 32, 32, 3)
```

```
X_test.shape
```

```
     (10000, 32, 32, 3)
```

```
y_train.shape
```

```
     (50000, 10)
```

```
y_test.shape
```

```
     (10000, 10)
```

```
def normalize_pixels(train, test):
    '''
    Normalize data into range of 0 to 1
    '''
    train_norm = train.astype('float32')
    test_norm  = test.astype('float32')

    train_norm /= 255
    test_norm /= 255

    return (train_norm, test_norm)
```

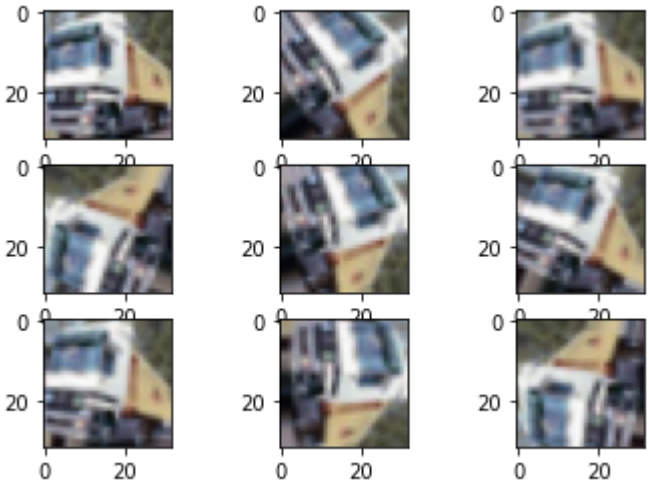
```
X_train,X_test=normalize_pixels(X_train,X_test)


#Reference: https://machinelearningmastery.com/how-to-configure-image-data-augmentation-wh
sample_image=X_train[1]
sample_image.shape
```

```
(32, 32, 3)
```

```
sample_images = expand_dims(sample_image, 0)
# create image data augmentation generator
datagen = ImageDataGenerator(rotation_range=90)
# prepare iterator
it = datagen.flow(sample_images, batch_size=1)
# generate sample images and plot
for i in range(9):
    pyplot.subplot(330 + 1 + i)
    # generate batch of images
    batch = it.next()
    image = batch[0];
    # plot raw pixel data
    pyplot.imshow(image)


pyplot.show()
```



1.1 Model using Dense Layer

```
#https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-c

def model_summarize(history):
    '''
    Summarize model i.e. print train and test loss
    '''
    # plot loss
    pyplot.subplot(121)
```

```
        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 model_harness(X_train, y_train, X_test, y_test, given_batch_size, given_step_size, giv
        '''
        define model using data augmentation technique and extend it to it's vertical limit
        '''
        # model = pickle.load('densenet.pkl')
        # create data generator
        datagen = ImageDataGenerator(width_shift_range=0.1, height_shift_range=0.1, horizontal

        # prepare iterator
        iterator_train = datagen.flow(X_train, y_train, batch_size=given_batch_size)

        # fit model
        steps = int(X_train.shape[0] / given_step_size)
        history = model.fit_generator(iterator_train, steps_per_epoch=steps, epochs=given_epoc

        # evaluate model
        _, acc = model.evaluate(X_test, y_test, verbose=1)
        print('> %.3f' % (acc * 100.0))
        # file = open('/densenet.pkl', 'wb')
        # pickle.dumps(model)#, file
        # learning curves
        model_summarize(history)


    def denseblock(input, num_filter = 64, dropout_rate = 0):

        #code for 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="
            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):

        #code for transition block

        global compression
        BatchNorm = layers.BatchNormalization()(input)
        relu = layers.Activation('relu')(BatchNorm)
```

```
        Conv2D_BottleNeck = layers.Conv2D(int(num_filter*compression), (5,5), kernel_initializ
        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):

        #Code for 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 = 10
    dropout_rate = 0
    l = 12
    input = layers.Input(shape=(img_height, img_width, channel))
    First_Conv2D = layers.Conv2D(num_filter, (5,5), use_bias=False ,padding='same')(input)
    BatchNorm = layers.BatchNormalization()(First_Conv2D)

    First_Block = denseblock(BatchNorm,32, dropout_rate)
    First_Transition = transition(First_Block, num_filter, dropout_rate)

    Second_Block = denseblock(First_Transition, 16, dropout_rate)
    Second_Transition = transition(Second_Block, num_filter, dropout_rate)

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

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


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

```
    _________________________________________________________________
    conv2d_97 (Conv2D)              (None, 4, 4, 5)      3755        activation_96[0][(
    _________________________________________________________________
    concatenate_89 (Concatenate)    (None, 4, 4, 35)     0           concatenate_88[0]
                                                                     conv2d_97[0][0]
    _________________________________________________________________
    batch_normalization_98 (BatchNo (None, 4, 4, 35)     140         concatenate_89[0]
    _________________________________________________________________
    activation_97 (Activation)      (None, 4, 4, 35)     0           batch_normalizati
    _________________________________________________________________
    conv2d_98 (Conv2D)              (None, 4, 4, 5)      4380        activation_97[0][(
    _________________________________________________________________
    concatenate_90 (Concatenate)    (None, 4, 4, 40)     0           concatenate_89[0]
                                                                     conv2d_98[0][0]
```

```
batch_normalization_99 (BatchNo (None, 4, 4, 40)     160         concatenate_90[0]
activation_98 (Activation)      (None, 4, 4, 40)     0           batch_normalizati
conv2d_99 (Conv2D)              (None, 4, 4, 5)      5005        activation_98[0][
concatenate_91 (Concatenate)    (None, 4, 4, 45)     0           concatenate_90[0]
                                                                 conv2d_99[0][0]
batch_normalization_100 (BatchN (None, 4, 4, 45)     180         concatenate_91[0]
activation_99 (Activation)      (None, 4, 4, 45)     0           batch_normalizati
conv2d_100 (Conv2D)             (None, 4, 4, 5)      5630        activation_99[0][
concatenate_92 (Concatenate)    (None, 4, 4, 50)     0           concatenate_91[0]
                                                                 conv2d_100[0][0]
batch_normalization_101 (BatchN (None, 4, 4, 50)     200         concatenate_92[0]
activation_100 (Activation)     (None, 4, 4, 50)     0           batch_normalizati
conv2d_101 (Conv2D)             (None, 4, 4, 5)      6255        activation_100[0]
concatenate_93 (Concatenate)    (None, 4, 4, 55)     0           concatenate_92[0]
                                                                 conv2d_101[0][0]
batch_normalization_102 (BatchN (None, 4, 4, 55)     220         concatenate_93[0]
activation_101 (Activation)     (None, 4, 4, 55)     0           batch_normalizati
conv2d_102 (Conv2D)             (None, 4, 4, 5)      6880        activation_101[0]
concatenate_94 (Concatenate)    (None, 4, 4, 60)     0           concatenate_93[0]
                                                                 conv2d_102[0][0]
batch_normalization_103 (BatchN (None, 4, 4, 60)     240         concatenate_94[0]
activation_102 (Activation)     (None, 4, 4, 60)     0           batch_normalizati
conv2d_103 (Conv2D)             (None, 4, 4, 5)      7505        activation_102[0]
concatenate_95 (Concatenate)    (None, 4, 4, 65)     0           concatenate_94[0]
```

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


# entry point, run the test harness
model_harness(X_train, y_train, X_test, y_test, 64, 64, 75)
```

```
Epoch 28/75
781/781 [==============================] - 83s 106ms/step - loss: 0.3571 - accuracy:
Epoch 29/75
781/781 [==============================] - 82s 106ms/step - loss: 0.3428 - accuracy:
Epoch 30/75
781/781 [==============================] - 81s 104ms/step - loss: 0.3372 - accuracy:
Epoch 31/75
781/781 [==============================] - 81s 104ms/step - loss: 0.3238 - accuracy:
Epoch 32/75
781/781 [==============================] - 81s 104ms/step - loss: 0.3243 - accuracy:
Epoch 33/75
781/781 [==============================] - 81s 104ms/step - loss: 0.3156 - accuracy:
Epoch 34/75
781/781 [==============================] - 81s 104ms/step - loss: 0.3077 - accuracy:
Epoch 35/75
781/781 [==============================] - 81s 104ms/step - loss: 0.3012 - accuracy:
Epoch 36/75
781/781 [==============================] - 81s 104ms/step - loss: 0.2919 - accuracy:
Epoch 37/75
781/781 [==============================] - 81s 104ms/step - loss: 0.2878 - accuracy:
Epoch 38/75
781/781 [==============================] - 81s 104ms/step - loss: 0.2846 - accuracy:
Epoch 39/75
781/781 [==============================] - 80s 103ms/step - loss: 0.2762 - accuracy:
Epoch 40/75
781/781 [==============================] - 81s 103ms/step - loss: 0.2674 - accuracy:
Epoch 41/75
781/781 [==============================] - 81s 103ms/step - loss: 0.2639 - accuracy:
Epoch 42/75
781/781 [==============================] - 81s 103ms/step - loss: 0.2562 - accuracy:
Epoch 43/75
781/781 [==============================] - 81s 103ms/step - loss: 0.2523 - accuracy:
Epoch 44/75
781/781 [==============================] - 81s 103ms/step - loss: 0.2468 - accuracy:
Epoch 45/75
781/781 [==============================] - 80s 103ms/step - loss: 0.2411 - accuracy:
Epoch 46/75
781/781 [==============================] - 81s 104ms/step - loss: 0.2379 - accuracy:
Epoch 47/75
781/781 [==============================] - 81s 104ms/step - loss: 0.2392 - accuracy:
Epoch 48/75
781/781 [==============================] - 82s 104ms/step - loss: 0.2299 - accuracy:
Epoch 49/75
781/781 [==============================] - 83s 107ms/step - loss: 0.2287 - accuracy:
Epoch 50/75
781/781 [==============================] - 83s 107ms/step - loss: 0.2158 - accuracy:
Epoch 51/75
781/781 [==============================] - 83s 106ms/step - loss: 0.2181 - accuracy:
Epoch 52/75
781/781 [==============================] - 82s 105ms/step - loss: 0.2116 - accuracy:
Epoch 53/75
781/781 [==============================] - 82s 104ms/step - loss: 0.2081 - accuracy:
Epoch 54/75
781/781 [==============================] - 82s 104ms/step - loss: 0.2055 - accuracy:
Epoch 55/75
781/781 [==============================] - 81s 104ms/step - loss: 0.1993 - accuracy:
Epoch 56/75
781/781 [==============================] - 81s 104ms/step - loss: 0.1996 - accuracy:
```

```
# Save the trained weights in to .h5 format
model.save_weights("DNST_model_with_dense_layer.h5")
print("Saved model to disk")
```

```
Saved model to disk
```

```
# free model variable
del model
```

2 Model without applying Dense layer and implemented all instructions suggested in the assignment

```
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 ,pad

        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 ,

    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.MaxPooling2D(pool_size=(2,2))(relu)
```

```
    AvgPooling = layers.MaxPooling2D(pool_size=(2,2))(relu)

    output = layers.Conv2D(filters=10,kernel_size=(2,2),activation='softmax')(AvgPooling)

    flat = layers.Flatten()(output)

    return flat


num_filter = 12
dropout_rate = 0
l = 12
input = layers.Input(shape=(img_height, img_width, channel,))
First_Conv2D = layers.Conv2D(32, (3,3), 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()
```

```
batch_normalization_62 (BatchNo (None, 32, 32, 77)   308         concatenate_56[0]
activation_61 (Activation)      (None, 32, 32, 77)   0           batch_normalizati
conv2d_62 (Conv2D)              (None, 32, 32, 5)    9625        activation_61[0][
concatenate_57 (Concatenate)    (None, 32, 32, 82)   0           concatenate_56[0]
                                                                 conv2d_62[0][0]
batch_normalization_63 (BatchNo (None, 32, 32, 82)   328         concatenate_57[0]
activation_62 (Activation)      (None, 32, 32, 82)   0           batch_normalizati
conv2d_63 (Conv2D)              (None, 32, 32, 5)    10250       activation_62[0][
concatenate_58 (Concatenate)    (None, 32, 32, 87)   0           concatenate_57[0]
                                                                 conv2d_63[0][0]
batch_normalization_64 (BatchNo (None, 32, 32, 87)   348         concatenate_58[0]
activation_63 (Activation)      (None, 32, 32, 87)   0           batch_normalizati
conv2d_64 (Conv2D)              (None, 32, 32, 5)    10875       activation_63[0][
concatenate_59 (Concatenate)    (None, 32, 32, 92)   0           concatenate_58[0]
                                                                 conv2d_64[0][0]
batch_normalization_65 (BatchNo (None, 32, 32, 92)   368         concatenate_59[0]
```

activation_64 (Activation)	(None, 32, 32, 92)	0	batch_normalizati
conv2d_65 (Conv2D)	(None, 32, 32, 32)	73600	activation_64[0][
average_pooling2d_4 (AveragePoo	(None, 16, 16, 32)	0	conv2d_65[0][0]
batch_normalization_66 (BatchNo	(None, 16, 16, 32)	128	average_pooling2d
activation_65 (Activation)	(None, 16, 16, 32)	0	batch_normalizati
conv2d_66 (Conv2D)	(None, 16, 16, 5)	4000	activation_65[0][
concatenate_60 (Concatenate)	(None, 16, 16, 37)	0	average_pooling2d conv2d_66[0][0]
batch_normalization_67 (BatchNo	(None, 16, 16, 37)	148	concatenate_60[0]
activation_66 (Activation)	(None, 16, 16, 37)	0	batch_normalizati
conv2d_67 (Conv2D)	(None, 16, 16, 5)	4625	activation_66[0][
concatenate_61 (Concatenate)	(None, 16, 16, 42)	0	concatenate_60[0] conv2d_67[0][0]
batch_normalization_68 (BatchNo	(None, 16, 16, 42)	168	concatenate_61[0]
activation_67 (Activation)	(None, 16, 16, 42)	0	batch_normalizati

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


# sample run
model_harness(X_train, y_train, X_test, y_test, 60, 39, 1)
```

```
WARNING:tensorflow:From <ipython-input-22-9641928ef676>:12: Model.fit_generator (from
Instructions for updating:
Please use Model.fit, which supports generators.
1282/1282 [==============================] - 125s 97ms/step - loss: 1.3586 - accuracy
```

```
# entry point, run the test harness for 50 * 5 iterations, 1st slot will·be:
model_harness(X_train, y_train, X_test, y_test, 60, 39, 50)
```

```
Epoch 1/50
1282/1282 [==============================] - 122s 95ms/step - loss: 0.8660 - accur
Epoch 2/50
1282/1282 [==============================] - 123s 96ms/step - loss: 0.6867 - accur
Epoch 3/50
1282/1282 [==============================] - 122s 95ms/step - loss: 0.5855 - accur
Epoch 4/50
1282/1282 [==============================] - 122s 95ms/step - loss: 0.5207 - accur
Epoch 5/50
1282/1282 [==============================] - 122s 95ms/step - loss: 0.4703 - accur
Epoch 6/50
1282/1282 [==============================] - 122s 95ms/step - loss: 0.4331 - accur
Epoch 7/50
1282/1282 [==============================] - 122s 95ms/step - loss: 0.3976 - accur
Epoch 8/50
1282/1282 [==============================] - 122s 96ms/step - loss: 0.3685 - accur
Epoch 9/50
1282/1282 [==============================] - 123s 96ms/step - loss: 0.3502 - accur
Epoch 10/50
1282/1282 [==============================] - 123s 96ms/step - loss: 0.3253 - accur
Epoch 11/50
1282/1282 [==============================] - 123s 96ms/step - loss: 0.3065 - accur
Epoch 12/50
1282/1282 [==============================] - 122s 95ms/step - loss: 0.2877 - accur
Epoch 13/50
1282/1282 [==============================] - 122s 95ms/step - loss: 0.2827 - accur
Epoch 14/50
1282/1282 [==============================] - 122s 95ms/step - loss: 0.2626 - accur
Epoch 15/50
1282/1282 [==============================] - 124s 97ms/step - loss: 0.2538 - accur
Epoch 16/50
1282/1282 [==============================] - 123s 96ms/step - loss: 0.2365 - accur
Epoch 17/50
1282/1282 [==============================] - 123s 96ms/step - loss: 0.2280 - accur
Epoch 18/50
1282/1282 [==============================] - 123s 96ms/step - loss: 0.2166 - accur
Epoch 19/50
1282/1282 [==============================] - 124s 96ms/step - loss: 0.2117 - accur
Epoch 20/50
1282/1282 [==============================] - 123s 96ms/step - loss: 0.2027 - accur
Epoch 21/50
1282/1282 [==============================] - 123s 96ms/step - loss: 0.1952 - accur
Epoch 22/50
1282/1282 [==============================] - 123s 96ms/step - loss: 0.1877 - accur
Epoch 23/50
1282/1282 [==============================] - 123s 96ms/step - loss: 0.1831 - accur
Epoch 24/50
1282/1282 [==============================] - 123s 96ms/step - loss: 0.1742 - accur
Epoch 25/50
1282/1282 [==============================] - 123s 96ms/step - loss: 0.1716 - accur
Epoch 26/50
```

```
1282/1282 [==============================] - 123s 96ms/step - loss: 0.1640 - accur
Epoch 27/50
1282/1282 [==============================] - 123s 96ms/step - loss: 0.1590 - accur
Epoch 28/50
1282/1282 [==============================] - 123s 96ms/step - loss: 0.1518 - accur
```

```
# entry point, run the test harness for 25 * 8 iterations, 2nd slot will be:
model_harness(X_train, y_train, X_test, y_test, 60, 39, 25)
```

```
Epoch 6/25
1282/1282 [==============================] - 121s 94ms/step - loss: 0.0812 - accuracy
Epoch 7/25
1282/1282 [==============================] - 121s 94ms/step - loss: 0.0774 - accuracy
Epoch 8/25
1282/1282 [==============================] - 121s 94ms/step - loss: 0.0793 - accuracy
Epoch 9/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0764 - accuracy
Epoch 10/25
1282/1282 [==============================] - 121s 94ms/step - loss: 0.0740 - accuracy
Epoch 11/25
1282/1282 [==============================] - 121s 94ms/step - loss: 0.0739 - accuracy
Epoch 12/25
1282/1282 [==============================] - 120s 94ms/step - loss: 0.0740 - accuracy
Epoch 13/25
1282/1282 [==============================] - 121s 94ms/step - loss: 0.0728 - accuracy
Epoch 14/25
1282/1282 [==============================] - 120s 94ms/step - loss: 0.0685 - accuracy
Epoch 15/25
```

```
# entry point, run the test harness for 25 * 8 iterations, 3rd slot will be:
model_harness(X_train, y_train, X_test, y_test, 60, 39, 25)
```

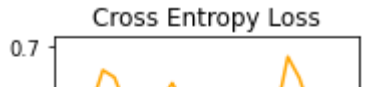
```
Epoch 1/25
1282/1282 [==============================] - 120s 94ms/step - loss: 0.0615 - accuracy
Epoch 2/25
1282/1282 [==============================] - 120s 94ms/step - loss: 0.0589 - accuracy
Epoch 3/25
1282/1282 [==============================] - 121s 95ms/step - loss: 0.0590 - accuracy
Epoch 4/25
1282/1282 [==============================] - 121s 94ms/step - loss: 0.0590 - accuracy
Epoch 5/25
1282/1282 [==============================] - 121s 94ms/step - loss: 0.0579 - accuracy
Epoch 6/25
1282/1282 [==============================] - 121s 94ms/step - loss: 0.0569 - accuracy
Epoch 7/25
1282/1282 [==============================] - 121s 95ms/step - loss: 0.0593 - accuracy
Epoch 8/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0556 - accuracy
Epoch 9/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0542 - accuracy
Epoch 10/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0527 - accuracy
Epoch 11/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0538 - accuracy
Epoch 12/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0559 - accuracy
Epoch 13/25
1282/1282 [==============================] - 123s 96ms/step - loss: 0.0528 - accuracy
Epoch 14/25
1282/1282 [==============================] - 123s 96ms/step - loss: 0.0529 - accuracy
Epoch 15/25
1282/1282 [==============================] - 123s 96ms/step - loss: 0.0509 - accuracy
Epoch 16/25
```

```
# entry point, run the test harness for 25 * 8 iterations, 4th slot will be:
model_harness(X_train, y_train, X_test, y_test, 60, 39, 25)
```

```
Epoch 1/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0470 - accuracy
Epoch 2/25
1282/1282 [==============================] - 121s 95ms/step - loss: 0.0469 - accuracy
Epoch 3/25
1282/1282 [==============================] - 121s 94ms/step - loss: 0.0456 - accuracy
Epoch 4/25
1282/1282 [==============================] - 121s 94ms/step - loss: 0.0457 - accuracy
Epoch 5/25
1282/1282 [==============================] - 120s 94ms/step - loss: 0.0422 - accuracy
Epoch 6/25
1282/1282 [==============================] - 121s 94ms/step - loss: 0.0474 - accuracy
Epoch 7/25
1282/1282 [==============================] - 121s 95ms/step - loss: 0.0458 - accuracy
Epoch 8/25
1282/1282 [==============================] - 121s 95ms/step - loss: 0.0423 - accuracy
Epoch 9/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0445 - accuracy
Epoch 10/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0422 - accuracy
Epoch 11/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0432 - accuracy
Epoch 12/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0420 - accuracy
Epoch 13/25
1282/1282 [==============================] - 122s 96ms/step - loss: 0.0425 - accuracy
Epoch 14/25
1282/1282 [==============================] - 123s 96ms/step - loss: 0.0410 - accuracy
Epoch 15/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0413 - accuracy
Epoch 16/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0426 - accuracy
Epoch 17/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0414 - accuracy
Epoch 18/25
1282/1282 [==============================] - 123s 96ms/step - loss: 0.0393 - accuracy
Epoch 19/25
1282/1282 [==============================] - 124s 96ms/step - loss: 0.0408 - accuracy
Epoch 20/25
1282/1282 [==============================] - 123s 96ms/step - loss: 0.0395 - accuracy
Epoch 21/25
1282/1282 [==============================] - 123s 96ms/step - loss: 0.0417 - accuracy
Epoch 22/25
```

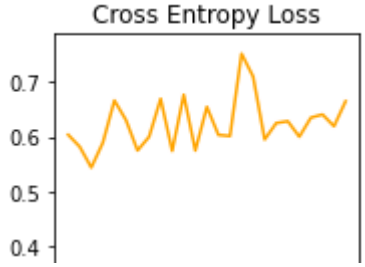
```
# entry point, run the test harness for 25 * 8 iterations, 5th slot will be:
model_harness(X_train, y_train, X_test, y_test, 60, 39, 25)
```

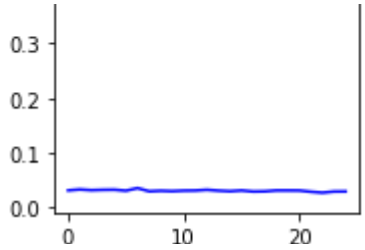
```
Epoch 1/25
1282/1282 [==============================] - 123s 96ms/step - loss: 0.0383 - accuracy
Epoch 2/25
1282/1282 [==============================] - 123s 96ms/step - loss: 0.0357 - accuracy
Epoch 3/25
1282/1282 [==============================] - 123s 96ms/step - loss: 0.0382 - accuracy
Epoch 4/25
1282/1282 [==============================] - 123s 96ms/step - loss: 0.0387 - accuracy
Epoch 5/25
1282/1282 [==============================] - 123s 96ms/step - loss: 0.0374 - accuracy
Epoch 6/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0356 - accuracy
Epoch 7/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0368 - accuracy
Epoch 8/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0361 - accuracy
Epoch 9/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0379 - accuracy
Epoch 10/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0323 - accuracy
Epoch 11/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0357 - accuracy
Epoch 12/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0369 - accuracy
Epoch 13/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0360 - accuracy
Epoch 14/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0362 - accuracy
Epoch 15/25
1282/1282 [==============================] - 123s 96ms/step - loss: 0.0345 - accuracy
Epoch 16/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0342 - accuracy
Epoch 17/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0370 - accuracy
Epoch 18/25
1282/1282 [==============================] - 123s 96ms/step - loss: 0.0315 - accuracy
Epoch 19/25
1282/1282 [==============================] - 123s 96ms/step - loss: 0.0347 - accuracy
Epoch 20/25
1282/1282 [==============================] - 123s 96ms/step - loss: 0.0333 - accuracy
Epoch 21/25
1282/1282 [==============================] - 124s 97ms/step - loss: 0.0347 - accuracy
Epoch 22/25
1282/1282 [==============================] - 123s 96ms/step - loss: 0.0340 - accuracy
Epoch 23/25
1282/1282 [==============================] - 124s 97ms/step - loss: 0.0330 - accuracy
Epoch 24/25
1282/1282 [==============================] - 123s 96ms/step - loss: 0.0326 - accuracy
Epoch 25/25
1282/1282 [==============================] - 123s 96ms/step - loss: 0.0323 - accuracy
313/313 [==============================] - 5s 17ms/step - loss: 0.5873 - accuracy: 0
> 89.190
```




```
# entry point, run the test harness for 25 * 8 iterations, 6th slot will be:
model_harness(X_train, y_train, X_test, y_test, 60, 39, 25)
```

```
Epoch 1/25
1282/1282 [==============================] - 123s 96ms/step - loss: 0.0309 - accuracy
Epoch 2/25
1282/1282 [==============================] - 123s 96ms/step - loss: 0.0327 - accuracy
Epoch 3/25
1282/1282 [==============================] - 122s 96ms/step - loss: 0.0313 - accuracy
Epoch 4/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0320 - accuracy
Epoch 5/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0322 - accuracy
Epoch 6/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0303 - accuracy
Epoch 7/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0351 - accuracy
Epoch 8/25
1282/1282 [==============================] - 121s 95ms/step - loss: 0.0293 - accuracy
Epoch 9/25
1282/1282 [==============================] - 121s 95ms/step - loss: 0.0302 - accuracy
Epoch 10/25
1282/1282 [==============================] - 121s 95ms/step - loss: 0.0295 - accuracy
Epoch 11/25
1282/1282 [==============================] - 121s 95ms/step - loss: 0.0304 - accuracy
Epoch 12/25
1282/1282 [==============================] - 121s 95ms/step - loss: 0.0307 - accuracy
Epoch 13/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0320 - accuracy
Epoch 14/25
1282/1282 [==============================] - 121s 94ms/step - loss: 0.0304 - accuracy
Epoch 15/25
1282/1282 [==============================] - 121s 95ms/step - loss: 0.0294 - accuracy
Epoch 16/25
1282/1282 [==============================] - 121s 94ms/step - loss: 0.0308 - accuracy
Epoch 17/25
1282/1282 [==============================] - 121s 94ms/step - loss: 0.0289 - accuracy
Epoch 18/25
1282/1282 [==============================] - 121s 94ms/step - loss: 0.0292 - accuracy
Epoch 19/25
1282/1282 [==============================] - 120s 94ms/step - loss: 0.0308 - accuracy
Epoch 20/25
1282/1282 [==============================] - 121s 94ms/step - loss: 0.0308 - accuracy
Epoch 21/25
1282/1282 [==============================] - 121s 94ms/step - loss: 0.0307 - accuracy
Epoch 22/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0284 - accuracy
Epoch 23/25
1282/1282 [==============================] - 121s 94ms/step - loss: 0.0267 - accuracy
Epoch 24/25
1282/1282 [==============================] - 121s 94ms/step - loss: 0.0288 - accuracy
Epoch 25/25
1282/1282 [==============================] - 121s 94ms/step - loss: 0.0290 - accuracy
313/313 [==============================] - 5s 16ms/step - loss: 0.6647 - accuracy: 0
> 88.550
```






```
# entry point, run the test harness for 25 * 8 iterations, 7th slot will be:
model_harness(X_train, y_train, X_test, y_test, 60, 39, 25)
```

```
Epoch 1/25
1282/1282 [==============================] - 121s 95ms/step - loss: 0.0280 - accuracy
Epoch 2/25
1282/1282 [==============================] - 121s 95ms/step - loss: 0.0294 - accuracy
Epoch 3/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0279 - accuracy
Epoch 4/25
1282/1282 [==============================] - 123s 96ms/step - loss: 0.0274 - accuracy
Epoch 5/25
1282/1282 [==============================] - 123s 96ms/step - loss: 0.0289 - accuracy
Epoch 6/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0293 - accuracy
Epoch 7/25
1282/1282 [==============================] - 121s 95ms/step - loss: 0.0287 - accuracy
Epoch 8/25
1282/1282 [==============================] - 121s 95ms/step - loss: 0.0271 - accuracy
Epoch 9/25
1282/1282 [==============================] - 121s 94ms/step - loss: 0.0285 - accuracy
Epoch 10/25
```

```
# entry point, run the test harness for 25 * 8 iterations, 8th slot will be:
model_harness(X_train, y_train, X_test, y_test, 60, 39, 25)
```

```
Epoch 6/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0245 - accuracy
Epoch 7/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0245 - accuracy
Epoch 8/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0266 - accuracy
Epoch 9/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0245 - accuracy
Epoch 10/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0224 - accuracy
Epoch 11/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0265 - accuracy
Epoch 12/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0235 - accuracy
Epoch 13/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0237 - accuracy
Epoch 14/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0262 - accuracy
Epoch 15/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0226 - accuracy
Epoch 16/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0238 - accuracy
Epoch 17/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0249 - accuracy
Epoch 18/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0214 - accuracy
Epoch 19/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0241 - accuracy
Epoch 20/25
1282/1282 [==============================] - 122s 95ms/step - loss: 0.0240 - accuracy
Epoch 21/25
```

```
# Save the trained weights in to .h5 format
model.save_weights("DNST_model_without_dense_layer.h5")
print("Saved model to disk")
```

```
Saved model to disk
```

```
## 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,
##it really saves the computing time as well.
```

```
# entry point, run the test harness
model_harness(X_train, y_train, X_test, y_test, 60, 39, 25)
```