CNN on CIFAR

About CIFAR dataset:

Dataset and consists of 60,000 32x32 color images containing one of 10 object classes, with 6000 images per class.

Assignment instructions

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.

> Reference: https://arxiv.org/abs/1608.06993 (https://arxiv.org/abs/1608.06993)

Labels in CIFAR-10 dataset

0: airplane 1: automobile 2: bird 3: cat 4: deer 5: dog 6: frog 7: horse 8: ship 9: truck

Importing libraries

In [1]:

```
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import Concatenate
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, RMSprop
from tensorflow.keras.callbacks import Callback, EarlyStopping,ModelCheckpoint
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt

from skimage import exposure
from skimage.filters import unsharp_mask
import cv2

from tensorflow.keras.initializers import he_normal
from time import time
from tensorflow.python.keras.callbacks import TensorBoard
```

Loading Data

In [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, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)
```

In [3]:

```
X_train,X_cv,y_train,y_cv = train_test_split(X_train,y_train,stratify = y_train,train_size
```

In [4]:

```
X_train.shape
```

Out[4]:

```
(40000, 32, 32, 3)
```

In [5]:

```
X_train[0].shape
```

Out[5]:

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

In [6]:

```
X_test.shape
```

Out[6]:

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

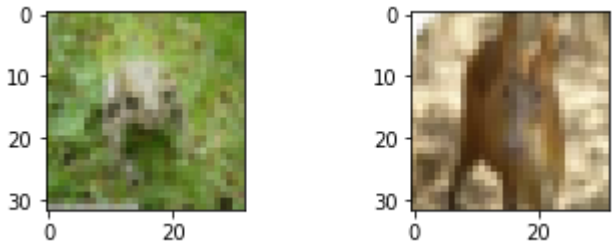
Sample images

In [7]:

```
#https://stackoverflow.com/questions/41793931/plotting-images-side-by-side-using-matplotlib
fig = plt.figure()
ax1 = fig.add_subplot(2,2,1)
ax1.imshow(X_train[4].reshape(32,32,3))
ax2 = fig.add_subplot(2,2,2)
ax2.imshow(X_train[7].reshape(32,32,3))
```

Out[7]:

```
<matplotlib.image.AxesImage at 0x24d9deb7550>
```



Augmenting Images

In [8]:

```
#https://keras.io/preprocessing/image/

from keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    zoom_range=0.5,
    brightness_range=[0.2,1.0])


datagen.fit(X_train)
```

```
Using TensorFlow backend.
```

Defining blocks - (Dense, Transition and Output_layer)

In [9]:

```
# Dense Block
def denseblock(input, num_filter, dropout_rate):
    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 ,padd
        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 Block
def transition(input, num_filter, dropout_rate):
    global compression
    BatchNorm = layers.BatchNormalization()(input)
    relu = layers.Activation('relu')(BatchNorm)
    Conv2D_BottleNeck = layers.Conv2D(int(num_filter*compression), (1,1), use_bias=False ,p
    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
```

Model

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	[1 × 1 conv, 3 × 3 conv] × 24	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 48	[1 × 1 conv, 3 × 3 conv] × 64
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	[1 × 1 conv, 3 × 3 conv] × 16	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 48
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

In [10]:

```
# Hyperparameters
batch_size = 64
num_classes = 10
epochs = 100
l = 5
compression = 0.8
dropout_rate = 0.2
num_filter = 32
```

In [11]:

```
input = layers.Input(shape=(img_height, img_width, channel))
First_Conv2D = layers.Conv2D(32, (3,3),strides=2, use_bias=False ,padding='same',activation

First_Block = denseblock(First_Conv2D, 32, 0.2)
First_Transition = transition(First_Block, 32, 0.2)


Second_Block = denseblock(First_Transition, 64, 0.2)
Second_Transition = transition(Second_Block, 64, 0.2)


Third_Block = denseblock(Second_Transition, 128, 0.2)
Third_Transition = transition(Third_Block, 128, 0.2)


Last_Block = denseblock(Third_Transition, 128, 0.2)
output = output_layer(Last_Block)

model = Model(inputs=[input], outputs=[output])
model.summary()
tensorboard = TensorBoard(log_dir="logs/".format(time))

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

```
[0][0]
__________________________________________________________________________________
_____________________
batch_normalization (BatchNorma (None, 16, 16, 32)   128         conv2d[0]
[0]
__________________________________________________________________________________
_____________________
activation (Activation)         (None, 16, 16, 32)   0           batch_nor
malization[0][0]
__________________________________________________________________________________
_____________________
conv2d_1 (Conv2D)               (None, 16, 16, 25)   7200        activatio
n[0][0]
__________________________________________________________________________________
_____________________
dropout (Dropout)               (None, 16, 16, 25)   0           conv2d_1
[0][0]
__________________________________________________________________________________
_____________________
concatenate (Concatenate)       (None, 16, 16, 57)   0           conv2d[0]
```

Creating model checkpoint

In [15]:

```
filepath="weights_1.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_auc', verbose=1, save_best_only=True, m

model.fit_generator(datagen.flow(X_train, y_train, batch_size=64), epochs=1,verbose=1,
                    validation_data=(X_cv, y_cv), callbacks=[tensorboard,checkpoint])
```

```
10000/625 [==============================================================
==============================================================================
==============================================================================
==============================================================================
==============================================================================
==============================================================================
==================================] - 4s 378us/sample - loss: 0.8673 - acc:
0.6371
WARNING:tensorflow:Can save best model only with val_auc available, skippin
g.
625/625 [==============================] - 75s 120ms/step - loss: 1.1465 - a
cc: 0.5956 - val_loss: 1.1391 - val_acc: 0.6371
```

Out[15]:

```
<tensorflow.python.keras.callbacks.History at 0x24d96e333c8>
```

In [ ]:

```
# Load TENSORBOARD
%load_ext tensorboard
# Start TENSORBOARD
%tensorboard --logdir logs --port=8000
```

In [16]:

```
model.load_weights("weights_1.hdf5")
model.compile(loss='categorical_crossentropy',
              optimizer = RMSprop(),
              metrics=['accuracy'])
```

```
---------------------------------------------------------------------------
OSError                                   Traceback (most recent call last)
<ipython-input-16-a0c3119ad891> in <module>
----> 1 model.load_weights("weights_1.hdf5")
      2 model.compile(loss='categorical_crossentropy',
      3               optimizer = RMSprop(),
      4               metrics=['accuracy'])

~\Anaconda3\envs\env\lib\site-packages\tensorflow_core\python\keras\engine\t
raining.py in load_weights(self, filepath, by_name)
    180         raise ValueError('Load weights is not yet supported with TPU
Strategy '
    181                          'with steps_per_run greater than 1.')
--> 182     return super(Model, self).load_weights(filepath, by_name)
    183
    184   @trackable.no_automatic_dependency_tracking

~\Anaconda3\envs\env\lib\site-packages\tensorflow_core\python\keras\engine\n
etwork.py in load_weights(self, filepath, by_name)
   1365           'first, then load the weights.')
   1366     self._assert_weights_created()
-> 1367     with h5py.File(filepath, 'r') as f:
   1368       if 'layer_names' not in f.attrs and 'model_weights' in f:
   1369         f = f['model_weights']

~\Anaconda3\envs\env\lib\site-packages\h5py\_hl\files.py in __init__(self, n
ame, mode, driver, libver, userblock_size, swmr, rdcc_nslots, rdcc_nbytes, r
dcc_w0, track_order, **kwds)
    392                 fid = make_fid(name, mode, userblock_size,
    393                                fapl, fcpl=make_fcpl(track_order=trac
k_order),
--> 394                                swmr=swmr)
    395
    396             if swmr_support:

~\Anaconda3\envs\env\lib\site-packages\h5py\_hl\files.py in make_fid(name, m
ode, userblock_size, fapl, fcpl, swmr)
    168         if swmr and swmr_support:
    169             flags |= h5f.ACC_SWMR_READ
--> 170         fid = h5f.open(name, flags, fapl=fapl)
    171     elif mode == 'r+':
    172         fid = h5f.open(name, h5f.ACC_RDWR, fapl=fapl)

h5py\_objects.pyx in h5py._objects.with_phil.wrapper()

h5py\_objects.pyx in h5py._objects.with_phil.wrapper()

h5py\h5f.pyx in h5py.h5f.open()

OSError: Unable to open file (unable to open file: name = 'weights_1.hdf5',
 errno = 2, error message = 'No such file or directory', flags = 0, o_flags
 = 0)
```

Evaluating model performance

In [ ]:

```
# Test the model
score = model.evaluate(X_test, y_test, verbose=1)
print("|","-"*50,"|")
print('  Test loss:', score[0])
print("|","-"*50,"|")
print('  Test accuracy:', score[1])
print("|","-"*50,"|")
```

In [ ]:

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