CNN on CIFAR :

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.

In [2]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n
4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_ty
pe=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.co
m%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%20https%3a%2f%2fwww.goog
leapis.com%2fauth%2fpeopleapi.readonly (https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6b
n6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&
response_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.go
ogleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%20https%3a%2
f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly)

Enter your authorization code:
··········
Mounted at /content/drive
```

In [0]:
```
# import keras
# from keras.datasets import cifar10
# from keras.models import Model, Sequential
# from keras.layers import Dense, Dropout, Flatten, Input, AveragePooling2D, merge, Activation
# from keras.layers import Conv2D, MaxPooling2D, BatchNormalization
# from keras.layers import Concatenate
# from keras.optimizers import Adam
import tensorflow as tf
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
```

In [0]:
```
# Hyperparameters
batch_size = 128
num_classes = 10
epochs = 30
l = 40
num_filter = 10
compression = 0.5
dropout_rate = 0
```

In [5]:
```
# 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 (https://www.cs.toronto.edu/~
kriz/cifar-10-python.tar.gz)
170500096/170498071 [==============================] - 11s 0us/step
```

In [6]:
```
X_train.shape
```

Out[6]:
```
(50000, 32, 32, 3)
```

In [7]:
```
X_test.shape
```
Out[7]:
```
(10000, 32, 32, 3)
```

In [0]:
```
#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 ,padding='same')(relu)
        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 ,padding='same')(re
    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
```

#Defining the model architecture

[illegible]

In [8]:

```
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)
```

```
WARNING:tensorflow:From /tensorflow-1.15.0/python3.6/tensorflow_core/python/ops/resource_variable_ops.py:16
30: calling BaseResourceVariable.__init__ (from tensorflow.python.ops.resource_variable_ops) with constrain
t is deprecated and will be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
```

In [9]:

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

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 32, 32, 3)]  0
__________________________________________________________________________________________________
conv2d (Conv2D)                 (None, 32, 32, 10)   270         input_1[0][0]
__________________________________________________________________________________________________
batch_normalization (BatchNorma (None, 32, 32, 10)   40          conv2d[0][0]
__________________________________________________________________________________________________
activation (Activation)         (None, 32, 32, 10)   0           batch_normalization[0][0]
__________________________________________________________________________________________________
conv2d_1 (Conv2D)               (None, 32, 32, 5)    450         activation[0][0]
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 32, 32, 15)   0           conv2d[0][0]
                                                                 conv2d_1[0][0]
__________________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 32, 32, 15)   60          concatenate[0][0]
__________________________________________________________________________________________________
```

#Standardizing the data

In [0]:

```
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

mean = X_train.mean(0)
dev = X_train.std(0)

def Standardization(data):
    data = data - mean
    data = data / dev
    return data

X_train = Standardization(X_train)
X_test = Standardization(X_test)
```

Data augementation

In [10]:
```
from keras.preprocessing.image import ImageDataGenerator
datagen_train = ImageDataGenerator(
    rotation_range=20,
        zoom_range=0.15,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.15,
    horizontal_flip=True,
)

datagen_train.fit(X_train)
```

```
Using TensorFlow backend.
```

Using checkpoint and early stopping method

In [0]:
```
from tensorflow.keras.callbacks import Callback, EarlyStopping, ModelCheckpoint,LearningRateScheduler
checkpoint_1 = ModelCheckpoint("densenet_model.hdf5",monitor="val_acc",mode="max",save_best_only = True,verb

earlystop_1 = EarlyStopping(monitor = 'val_acc',
                            mode="max",
                            min_delta = 0,
                            patience = 10,
                            verbose = 1,)
callbacks_1 = [earlystop_1,checkpoint_1]
```

In [0]:
```
# determine Loss function and Optimizer
model.compile(loss='categorical_crossentropy',
              optimizer=Adam(),
              metrics=['accuracy'])
```

#Suppressing warnings

In [0]:

```
def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn
```

In [0]:

```
history = model.fit_generator(datagen_train.flow(X_train, y_train, batch_size=batch_size),steps_per_epoch=(l
    epochs=epochs,
    verbose = 1,
    validation_data=(X_test, y_test),
    callbacks = callbacks_1
)
```

```
Epoch 1/30
1953/1953 [===========================>.] - ETA: 0s - loss: 0.2051 - acc: 0.9277Epoch 1/30
10000/1953 [==========================================================================================================================================================] - 8s 793us/sample - loss: 0.6152 - acc: 0.8753

Epoch 00001: val_acc did not improve from 0.87690
1954/1953 [==============================] - 744s 381ms/step - loss: 0.2051 - acc: 0.9277 - val_loss: 0.4402 - val_acc: 0.8753
Epoch 2/30
1953/1953 [===========================>.] - ETA: 0s - loss: 0.2026 - acc: 0.9286Epoch 1/30
10000/1953 [==========================================================================================================================================================] - 8s 805us/sample - loss: 0.4845 - acc: 0.8900

Epoch 00002: val_acc improved from 0.87690 to 0.89000, saving model to densenet_model.hdf5
1954/1953 [==============================] - 747s 382ms/step - loss: 0.2026 - acc: 0.9286 - val_loss: 0.3780 - val_acc: 0.8900
Epoch 3/30
```

#Loading the saved model

In [13]:
```
from numpy import loadtxt
from tensorflow.keras.models import load_model

# load model
model = load_model('/content/drive/My Drive/densenet_model.hdf5')
```

```
WARNING:tensorflow:From /tensorflow-1.15.0/python3.6/tensorflow_core/python/ops/init_ops.py:97: calling Glo
rotUniform.__init__ (from tensorflow.python.ops.init_ops) with dtype is deprecated and will be removed in a
future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the constructor
WARNING:tensorflow:From /tensorflow-1.15.0/python3.6/tensorflow_core/python/ops/init_ops.py:97: calling Zer
os.__init__ (from tensorflow.python.ops.init_ops) with dtype is deprecated and will be removed in a future
version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the constructor
WARNING:tensorflow:From /tensorflow-1.15.0/python3.6/tensorflow_core/python/ops/init_ops.py:97: calling One
s.__init__ (from tensorflow.python.ops.init_ops) with dtype is deprecated and will be removed in a future v
ersion.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the constructor
WARNING:tensorflow:From /tensorflow-1.15.0/python3.6/tensorflow_core/python/ops/resource_variable_ops.py:16
30: calling BaseResourceVariable.__init__ (from tensorflow.python.ops.resource_variable_ops) with constrain
t is deprecated and will be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
```

Test the model

In [14]:
```

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

```
10000/10000 [==============================] - 23s 2ms/sample - loss: 0.3354 - acc: 0.9004
Test loss: 0.3354230671226978
Test accuracy: 0.9004
```

Save the trained weights in to .h5 format

In [0]:
```
model.save_weights("Densenet_model_final.h5")
```

#Note:

1.I have acheieved a Test accuracy of 90.04%

2.The number of epochs I specified was 30 so to complete each epoch it took around 13 minutes so for 30 epochs it took 6 hours 30 minutes and i could reach a validation accuracy of 87.69%

3.I again reran the same cell and the epoch continued with test accuracy from 87.69% and after 14 more epochs it gave an test accuracy of 90.04% and following 3 epochs had reduced test accuracy and the best weights were saved and downloaded and I loaded it again and performed testing in another cell and gave test accuracy of 90.04%.

4.17 epochs ran during 2nd time and it took around 3 hours 40 mins to compute.

5.The number of total parameters are 0.8 million.

6.No dropouts or fully connected layers are used in architecture.

7.So in total I ran 47 epochs and the model was trained for 10 hours and 10 minutes after which google colab runtime got disconnected and i lost my variables thats the reason you can only see 17 epochs in output.