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

In [0]:
```
# Hyperparameters
batch_size = 128
num_classes = 10
epochs = 10
l = 40
num_filter = 12
compression = 0.5
#dropout_rate = 0.2
weight_decay = 1e-4
```

In [0]:
```
# Load CIFAR10 Data
from sklearn.model_selection import train_test_split
#from sklearn.utils import resample
(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

# X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.
y_train = tf.keras.utils.to_categorical(y_train, num_classes)
y_test = tf.keras.utils.to_categorical(y_test, num_classes)
# y_cv = tf.keras.utils.to_categorical(y_cv, num_classes)
```

In [35]:
```
X_train.shape
```

Out[35]:
```
(50000, 32, 32, 3)
```

In [0]:
```
# X_cv=0
```

In [37]:
```
X_test.shape
```

Out[37]:
```
(10000, 32, 32, 3)
```

In [0]:
```
# Dense Block
#https://machinelearningmastery.com/weight-regularization-to-reduce-overfitting-o
def denseblock(input, num_filter = 12, dropout_rate = 0.2):
    global compression
    global weight_decay
    temp = input
    input_conv1 = num_filter*4
    for _ in range(l):
        BatchNorm = layers.BatchNormalization(beta_regularizer=regularizers.l2(we
        relu = layers.Activation('relu')(BatchNorm)
        Conv2D_3_3 = layers.Conv2D(int(input_conv1),1, padding='same',kernel_ini

        BatchNorm = layers.BatchNormalization(beta_regularizer=regularizers.l2(we
        relu = layers.Activation('relu')(BatchNorm)
        Conv2D_3_4 = layers.Conv2D(int(num_filter),3, padding='same',kernel_init

        concat = layers.Concatenate()([temp,Conv2D_3_4])

        temp = concat

    return temp


## transition Blosck
#https://arthurdouillard.com/post/densenet/
def transition(input, num_filter = 12, dropout_rate = 0.2):

    global compression
    global weight_decay
    channel = input.shape.as_list()[-1]
    input_conv = channel*compression
    BatchNorm = layers.BatchNormalization(beta_regularizer=regularizers.l2(weigh
    relu = layers.Activation('relu')(BatchNorm)
    Conv2D_BottleNeck = layers.Conv2D(int(input_conv), 1,use_bias=False ,padding
    Conv2D_BottleNeck = layers.Activation('relu')(Conv2D_BottleNeck)
    avg = layers.AveragePooling2D(pool_size=(2,2))(Conv2D_BottleNeck)
    return avg


# #output layer
def output_layer(input):
    input_conv = num_filter*1
    BatchNorm = BatchNormalization(beta_regularizer=regularizers.l2(weight_decay
    relu = Activation('relu')(BatchNorm)
    AvgPooling = AveragePooling2D(pool_size=(8,8))(relu)
    #output = layers.Conv2D(10,1,use_bias=False ,padding='same',kernel_regulariz
    #flat = Flatten()(AvgPooling)
    output = layers.Conv2D(10,1,use_bias=False ,padding='same',kernel_regularize
    output = Activation('softmax')(output)
    #output = AveragePooling2D(pool_size=(1,1))(output)
    output = Flatten()(output)
    #output  = BatchNormalization(beta_regularizer=regularizers.l2(weight_decay),
   # output = Flatten()(output)
   # output = layers.Conv2D(num_classes,2, activation='softmax',kernel_regulariz
   # output = layers.Dense(num_classes, activation='softmax')(flat)
    return output
```

In [0]:
```
from tensorflow.keras import regularizers
num_filter = 12
dropout_rate = 0.2
l = 16
input = layers.Input(shape=(img_height, img_width, channel,))
First_Conv2D = layers.Conv2D(2*num_filter, 3, use_bias=False ,padding='same',bia

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)

output = output_layer(Third_Block)
```

In [40]:
```
model1 = Model(inputs=[input], outputs=[output])
model1.summary()
```

```
Model: "model_2"
____________________________________________________________________________
______________________
Layer (type)                    Output Shape         Param #     Connected to
============================================================================
=====================
input_2 (InputLayer)            [(None, 32, 32, 3)]  0
____________________________________________________________________________
______________________
conv2d_100 (Conv2D)             (None, 32, 32, 24)   648         input_2[0]
[0]
____________________________________________________________________________
______________________
batch_normalization_99 (BatchNo (None, 32, 32, 24)   96          conv2d_100
[0][0]
____________________________________________________________________________
______________________
activation_102 (Activation)     (None, 32, 32, 24)   0           batch_normal
ization_99[0][0]
```

In [0]:
```
#https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augme
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(
    zoom_range=0.32,
    rotation_range=18,
 #   height_shift_range=0.1,
#     width_shift_range=0.1,
    horizontal_flip=True,
   # vertical_flip=True,
    rescale=1./255,
    fill_mode='nearest')

cv_datagen = ImageDataGenerator(
  #  zoom_range=0.32,
  #  rotation_range=18,
 #   height_shift_range=0.1,
#     width_shift_range=0.1,
  #  horizontal_flip=True,
   # vertical_flip=True,
    rescale=1./255,
#    fill_mode='nearest'
    )
test_datagen = ImageDataGenerator(
   # zoom_range=0.32,
 #   rotation_range=18,
 #   height_shift_range=0.1,
#     width_shift_range=0.1,
  #  horizontal_flip=True,
   # vertical_flip=True,
    rescale=1./255,
   # fill_mode='nearest'
    )
```

In [0]:
```
train_datagen.fit(X_train)
# cv_datagen.fit(X_cv)
test_datagen.fit(X_test)
```

In [0]:
```
#https://machinelearningmastery.com/how-to-control-the-speed-and-stability-of-tra
epochs = 100
batch_size = 64
val_batch_size = 64
steps_per_epoch= len(y_train)//batch_size
validation_steps = len(y_test)//val_batch_size
```

In [0]:
```
from tensorflow.keras.callbacks import ModelCheckpoint
filepath = "/content/drive/My Drive/saved-model-{epoch:02d}-{val_acc:.2f}.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_acc', verbose=1, save_best_on
```

In [0]:
```
model1 = Model(inputs=[input], outputs=[output])


model1.compile(loss='categorical_crossentropy',
              optimizer=tf.keras.optimizers.Nadam(lr=0.001,beta_1=0.9, beta_2=0.
              metrics=['acc'])
```

In [46]:
```
history = model1.fit_generator(train_datagen.flow(X_train, y_train ,batch_size=1
 ),steps_per_epoch=steps_per_epoch,
                    epochs=80,
                    validation_data=test_datagen.flow(X_test,y_test,batch_size=1
```

```
Epoch 1/80
780/781 [============================>.] - ETA: 0s - loss: 1.7130 - acc: 0.57
27Epoch 1/80
156/781 [====>.........................] - ETA: 1:28 - loss: 2.0870 - acc: 0.
4856
Epoch 00001: val_acc improved from -inf to 0.48565, saving model to /content/
drive/My Drive/saved-model-01-0.49.hdf5
781/781 [==============================] - 649s 831ms/step - loss: 1.7126 - a
cc: 0.5728 - val_loss: 2.0870 - val_acc: 0.4856
Epoch 2/80
780/781 [============================>.] - ETA: 0s - loss: 1.1634 - acc: 0.73
65Epoch 1/80
156/781 [====>.........................] - ETA: 1:08 - loss: 1.0995 - acc: 0.
7532
Epoch 00002: val_acc improved from 0.48565 to 0.75322, saving model to /conte
nt/drive/My Drive/saved-model-02-0.75.hdf5
781/781 [==============================] - 388s 496ms/step - loss: 1.1632 - a
cc: 0.7365 - val_loss: 1.0995 - val_acc: 0.7532
Epoch 3/80
```

In [0]:
```
from keras.models import load_model
best_model = tf.keras.models.load_model('/content/drive/My Drive/saved-model-34-
```

From epoch 34 , accuracy value of cv data = 0.9019. So test model is evalutated based on this model value.

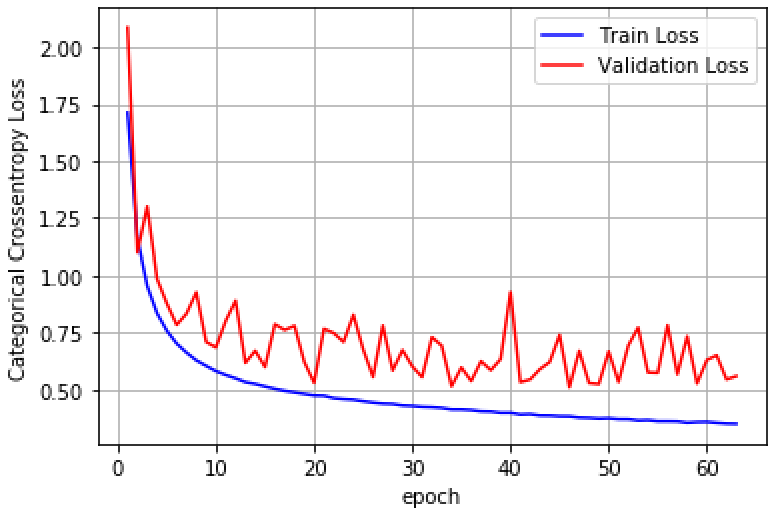
In [48]:
```
# evaluate on test data
score1 = best_model.evaluate_generator(test_datagen.flow(X_test, y_test, batch_s
```

```
157/157 [==============================] - 15s 95ms/step - loss: 0.5133 - acc:
0.9019
```

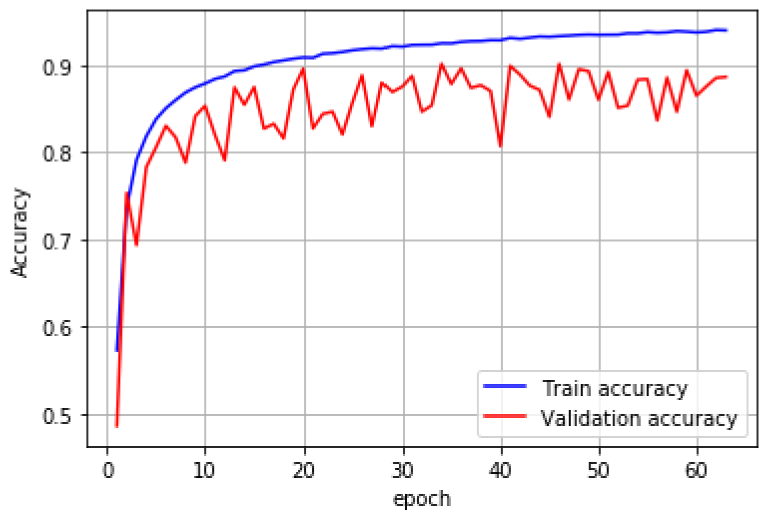
In [49]:
```
score1
```

Out[49]:
```
[0.5132799552884072, 0.9019]
```

In [8]:
```
import pylab as plt
def plt_dynamic(x, vy, ty, ax, colors=['b']):
 ax.plot(x, vy, 'b', label="Train Loss")
 ax.plot(x, ty, 'r', label="Validation Loss")
 plt.legend()
 plt.grid()
 fig.canvas.draw()
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')
# list of epoch numbers
x = list(range(1,64))
vy = history.history['loss']
ty = history.history['val_loss']
plt_dynamic(x, vy, ty, ax)
```



In [6]:
```
import pylab as plt
def plt_dynamic(x, vy, ty, ax, colors=['b']):
 ax.plot(x, vy, 'b', label="Train accuracy")
 ax.plot(x, ty, 'r', label="Validation accuracy")
 plt.legend()
 plt.grid()
 fig.canvas.draw()
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Accuracy')
# list of epoch numbers
x = list(range(1,64))
vy = history.history['acc']
ty = history.history['val_acc']
plt_dynamic(x, vy, ty, ax)
```



In [11]:
```
print("Test Accuracy {}". format(score1[1]))
```

```
Test Accuracy [90.194]
```

Test Accuracy = 0.9019