```
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, Nadam
import numpy as np
from tqdm import tqdm
from matplotlib import pyplot
from prettytable import PrettyTable
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.callbacks import ModelCheckpoint,LearningRateScheduler,CSVLogger, Callback,R
import matplotlib.pyplot as plt
from keras import models
import tensorflow as tf
```

```
The default version of TensorFlow in Colab will soon switch to TensorFlow 2.x.
We recommend you upgrade now or ensure your notebook will continue to use TensorFlow 1.x via the %te
Using TensorFlow backend.
```

```
final_tab = PrettyTable(['Augmentation','l','num_filters','compression','Optimizer','Te
```

```
# Hyperparameters
batch_size = 128
num_classes = 10
epochs = 100
l = 9
num_filter = 24
compression = 1.041
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)
```

```
print('Train Shape:',X_train.shape)
print('Test Shape:',X_test.shape)
```

```
# 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 ,
        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=Fals
    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
    print('input',input.shape)
    BatchNorm = layers.BatchNormalization()(input)
    print('Batch',BatchNorm.shape)
    relu = layers.Activation('relu')(BatchNorm)
    print('relu',relu.shape)
    AvgPooling = layers.AveragePooling2D(pool_size=(2,2))(relu)


    conv_layer = layers.Conv2D(10, (1,1), use_bias=False ,padding='same')(AvgPooling)
    last = layers.GlobalMaxPooling2D()(conv_layer)
    output = layers.Activation('softmax')(last)

    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_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()
```

```
# 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))
```

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

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

```
# ['Augmentation','l','num_filters','compression','Optimizer','Test Accuracy']

final_tab.add_row([None,l, num_filter, compression,'Adam',0.59])

```

```
print(final_tab)
```

DenseNet Function

```
def dense_net(xtrain,xtest, optim = Adam(),k_size=(3,3), b_size = batch_size, epoch = e
          print('b_size:{} epochs:{}'.format(b_size,epoch))
          input = layers.Input(shape=(img_height, img_width, channel,))
          First_Conv2D = layers.Conv2D(num_filter, (3,3), use_bias=False ,padding='same

          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)


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

          model.compile(loss='categorical_crossentropy',
```

```
                      optimizer=Adam(),
                      metrics=['accuracy'])

        model.fit(xtrain, y_train,
                            batch_size=batch_size,
                            epochs=epochs,
                            verbose=1,
                            validation_data=(xtest, y_test))




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

        return model


```

Image Augmentation Techniques

`Some of the augmentation techniques are as follows`

1. Vertical Shift Augmentation
2. Horizontal Shift Augmentation
3. Vertical Flip Augmentation
4. Horizontal Flip Augmentation

Vertical and Horizontal Shift Augmentation:

`A shift to an image means moving all pixels of the image in one direction, vertically,h dimensions the same.`

```
# Reff https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when

def vertical_horizontal_shift(arr_imgs):
      d = arr_imgs.copy()

      for i in tqdm(range(d.shape[0]), position=0):
          data = d[i]
          samples = expand_dims(data, 0)
          datagen = ImageDataGenerator(width_shift_range=[-15,15], height_shift_range=[
          it = datagen.flow(samples, batch_size=1)
          for j in range(9):
              batch = it.next()
              if j == 0:
                  image = batch[0].astype('uint8')
                  d[i] = image
                  break
```

```
17      return d
```

Original Image

```
1 pyplot.imshow(X_test[0])
```

After Vertical and Horizontal Shift

```
1 pyplot.imshow(vertical_horizontal_shift(X_test)[0])
```

Applying vertical and horizontal shift on vertical and horizontal shift

```
1 v_h_shift_train = vertical_horizontal_shift(X_train)
2 v_h_shift_test  = vertical_horizontal_shift(X_test)
3
```

```
1 pyplot.imshow(X_test[12])
```

```
1 pyplot.imshow(v_h_shift_test[12])
```

DenseNet with Adam Optimizer on Vertical Horizantal Shift Data

```
1 v_h_shift_model = dense_net(v_h_shift_train, v_h_shift_test)
```

```
final_tab.add_row(['Vertical_Horizantal_Shift',l, num_filter, compression,'Adam',0.42])
```

DenseNet with Nadam Optimizer on Vertical Horizantal Shift Data

```
v_h_shift_model_nadam = dense_net(v_h_shift_train, v_h_shift_test, optim=Nadam())
```

```
final_tab.add_row(['Vertical_Horizantal_Shift',l, num_filter, compression,'Nadam',0.41]
```

Horizontal and Vertical Flip Augmentation

```
An image flip means reversing the rows or columns of pixels in the case of a vertical o
```

```
# Reff https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when
def vertical_horizontal_flip(arr_imgs):

      d = arr_imgs.copy()

      for i in tqdm(range(d.shape[0])):
```

```
    for i in tqdm(range(d.shape[0])):
        data = d[i]
        samples = expand_dims(data, 0)
        datagen = ImageDataGenerator(vertical_flip=True, horizontal_flip=True)
        it = datagen.flow(samples, batch_size=1)
        for j in range(9):
          batch = it.next()
          if j == 2:
            image = batch[0].astype('uint8')
            d[i] = image
            break
    return d
```

```
1
```

DenseNet with Optimizer on Vertical Horizantal Flip Data

```
v_h_flip_xtrain = vertical_horizontal_flip(X_train)
v_h_flip_xtest  = vertical_horizontal_flip(X_test)
```

Before Flipping

```
pyplot.imshow(X_train[2])
```

After Flipping

```
pyplot.imshow(v_h_flip_xtrain[2])
```

```
v_h_flip_model = dense_net(v_h_flip_xtrain, v_h_flip_xtest)
```

```
final_tab.add_row(['Vertical_Horizantal_Flip',l, num_filter, compression,'Adam',0.55])

```

DenseNet with Nadam Optimizer on Vertical Horizantal Flip Data

```
v_h_flip_model_nadam = dense_net(v_h_flip_xtrain, v_h_flip_xtest, optim = Nadam())
```

```
final_tab.add_row(['Vertical_Horizantal_Flip',l, num_filter, compression,'Nadam',0.59])
```

Brightness Augmentation

```
The brightness of the image can be augmented by either randomly darkening images, brigh
```

```
def brightness(arr_imgs):
      d = arr_imgs.copy()

      for i in tqdm(range(d.shape[0])):
          data = d[i]
          samples = expand_dims(data, 0)
          datagen = ImageDataGenerator(brightness_range=[0.5,0.6])
          it = datagen.flow(samples, batch_size=1)
          for j in range(9):
            batch = it.next()
            if j == 8:
              image = batch[0].astype('uint8')
              d[i] = image
              break
      return d
```

```
bright_xtrain = brightness(X_train)
bright_xtest = brightness(X_test)
```

```
pyplot.imshow(X_train[2])
```

```
1 pyplot.imshow(bright_xtrain[2])
```

DenseNet with Adam Optimizer on Brightness Augmentation Data

```
1 bright_model = dense_net(bright_xtrain, bright_xtest)
```

```
1 final_tab.add_row(['Brightness',l, num_filter, compression,'Adam',0.67])
```

DenseNet with Nadam Optimizer on Brightness Augmentation Data

```
bright_model_nadam = dense_net(bright_xtrain, bright_xtest, optim=Nadam())
```

```
final_tab.add_row(['Brightness',l, num_filter, compression,'Nadam',0.67])
```

Feature Standardization

```
def standard(arr_imgs):

      d = arr_imgs.copy()

      for i in tqdm(range(d.shape[0])):
          data = d[i]
          samples = expand_dims(data, 0)
          datagen = ImageDataGenerator(featurewise_center=True, featurewise_std_normali
          it = datagen.flow(samples, batch_size=1)
          for j in range(9):
              batch = it.next()
              if j == 5:

                image = batch[0].astype('uint8')
                d[i] = image
                break
                # plot raw pixel data
      return d
```

```
# stand_xtrain = standard(X_train)
stand_xtest  = standard(X_test)
```

DenseNet with Adam Optimizer on Standardized Data

```
stand_model = dense_net(stand_xtrain, stand_xtest)
```

```
final_tab.add_row(['Standardized',l, num_filter, compression,'Adam',0.63])
```

DenseNet with Nadam Optimizer on Standardized Data

```
stand_model_nadam = dense_net(stand_xtrain,stand_xtest, optim = Nadam())
```

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/10
50000/50000 [==============================] - 117s 2ms/sample - loss: 1.7193 - acc:
Epoch 2/10
50000/50000 [==============================] - 98s 2ms/sample - loss: 1.3688 - acc: 0
Epoch 3/10
50000/50000 [==============================] - 98s 2ms/sample - loss: 1.2211 - acc: 0
Epoch 4/10
50000/50000 [==============================] - 98s 2ms/sample - loss: 1.1158 - acc: 0
Epoch 5/10
50000/50000 [==============================] - 98s 2ms/sample - loss: 1.0399 - acc: 0
Epoch 6/10
50000/50000 [==============================] - 98s 2ms/sample - loss: 0.9917 - acc: 0
Epoch 7/10
50000/50000 [==============================] - 98s 2ms/sample - loss: 0.9490 - acc: 0
Epoch 8/10
50000/50000 [==============================] - 98s 2ms/sample - loss: 0.9109 - acc: 0
Epoch 9/10
50000/50000 [==============================] - 98s 2ms/sample - loss: 0.8778 - acc: 0
Epoch 10/10
50000/50000 [==============================] - 98s 2ms/sample - loss: 0.8548 - acc: 0
10000/10000 [==============================] - 10s 1ms/sample - loss: 0.9976 - acc: 0
Test loss: 0.9975716045379639
Test accuracy: 0.6743
```

```
final_tab.add_row(['Standardized',l, num_filter, compression,'Nadam',0.63])
```

Now lets try with changing some of the parameters

```
l = 8
num_filter = 38
compression = 1
```

DenseNet with Adam Optimizer on Vertical Horizantal Shift

```
v_h_shift_model2 = dense_net(v_h_shift_train,v_h_shift_test)
```

```
final_tab.add_row(['Vertical_Horizantal_shift',l, num_filter, compression,'Adam',0.52])
```

DenseNet with Nadam Optimizer on Vertical Horizantal Shift

```
v_h_shift_model2_nadam = dense_net(v_h_shift_train,v_h_shift_test,optim=Nadam())
```

```
final_tab.add_row(['Vertical_Horizantal_shift',l, num_filter, compression,'Nadam',0.53]
```

DenseNet with Adam Optimizer on Vertical Horizantal Flip

```
v_h_flip_model2 = dense_net(v_h_flip_xtrain,v_h_flip_xtest)
```

```
final_tab.add_row(['Vertical_Horizantal_flip',l, num_filter, compression,'Adam',0.72])
```

DenseNet with Nadam Optimizer on Vertical Horizantal Flip

```
v_h_flip_model2_nadam = dense_net(v_h_flip_xtrain,v_h_flip_xtest,optim=Nadam())
```

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/10
50000/50000 [==============================] - 293s 6ms/sample - loss: 1.5328 - acc:
Epoch 2/10
50000/50000 [==============================] - 272s 5ms/sample - loss: 1.0911 - acc:
Epoch 3/10
50000/50000 [==============================] - 272s 5ms/sample - loss: 0.9227 - acc:
Epoch 4/10
50000/50000 [==============================] - 272s 5ms/sample - loss: 0.8073 - acc:
Epoch 5/10
50000/50000 [==============================] - 272s 5ms/sample - loss: 0.7319 - acc:
Epoch 6/10
50000/50000 [==============================] - 272s 5ms/sample - loss: 0.6665 - acc:
Epoch 7/10
50000/50000 [==============================] - 272s 5ms/sample - loss: 0.6127 - acc:
Epoch 8/10
50000/50000 [==============================] - 272s 5ms/sample - loss: 0.5698 - acc:
Epoch 9/10
50000/50000 [==============================] - 271s 5ms/sample - loss: 0.5264 - acc:
Epoch 10/10
50000/50000 [==============================] - 271s 5ms/sample - loss: 0.4910 - acc:
10000/10000 [==============================] - 18s 2ms/sample - loss: 1.1797 - acc: 0
Test loss: 1.1797264897346496
Test accuracy: 0.6763
```

```
final_tab.add_row(['Vertical_Horizantal_flip',l, num_filter, compression,'Nadam',0.67])
```

DenseNet with Adam Optimizer on Brightness

```
bright_model2 = dense_net(bright_xtrain, bright_xtest)
```

```
final_tab.add_row(['Brightness',l, num_filter, compression,'Adam',0.79])
```

DenseNet with Nadam Optimizer on Brightness

```
bright_model2_nadam = dense_net(bright_xtrain, bright_xtest, optim=Nadam())
```

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/10
50000/50000 [==============================] - 252s 5ms/sample - loss: 1.3535 - acc:
Epoch 2/10
50000/50000 [==============================] - 247s 5ms/sample - loss: 0.8746 - acc:
Epoch 3/10
50000/50000 [==============================] - 247s 5ms/sample - loss: 0.7027 - acc:
Epoch 4/10
50000/50000 [==============================] - 247s 5ms/sample - loss: 0.6028 - acc:
Epoch 5/10
50000/50000 [==============================] - 247s 5ms/sample - loss: 0.5348 - acc:
Epoch 6/10
50000/50000 [==============================] - 246s 5ms/sample - loss: 0.4797 - acc:
Epoch 7/10
50000/50000 [==============================] - 246s 5ms/sample - loss: 0.4343 - acc:
Epoch 8/10
50000/50000 [==============================] - 246s 5ms/sample - loss: 0.4059 - acc:
Epoch 9/10
50000/50000 [==============================] - 246s 5ms/sample - loss: 0.3660 - acc:
Epoch 10/10
50000/50000 [==============================] - 247s 5ms/sample - loss: 0.3360 - acc:
10000/10000 [==============================] - 15s 1ms/sample - loss: 1.2232 - acc: 0
Test loss: 1.2232139734268188
Test accuracy: 0.7018
```

```
final_tab.add_row(['Brightness',l, num_filter, compression,'Nadam',0.70])
```

DenseNet with Adam Optimizer on Standardized Data

```
stand_model2 = dense_net(stand_xtrain,stand_xtest)
```

```
final_tab.add_row(['Standardized',l, num_filter, compression,'Adam',0.77])
```

DenseNet with Nadam Optimizer on Standardized Data

```
stand_model2_nadam = dense_net(stand_xtrain,stand_xtest, optim = Nadam())
```

```
final_tab.add_row(['Standardized',l, num_filter, compression,'Nadam',0.80])
```

```
print(final_tab)
```

```
%%time

datagen = ImageDataGenerator(

    brightness_range=[0.5,1.9],
    featurewise_center=True, featurewise_std_normalization=True,
    width_shift_range = 0.125,
    horizontal_flip=True,vertical_flip=True,rotation_range=15,
    fill_mode='nearest'
)
```

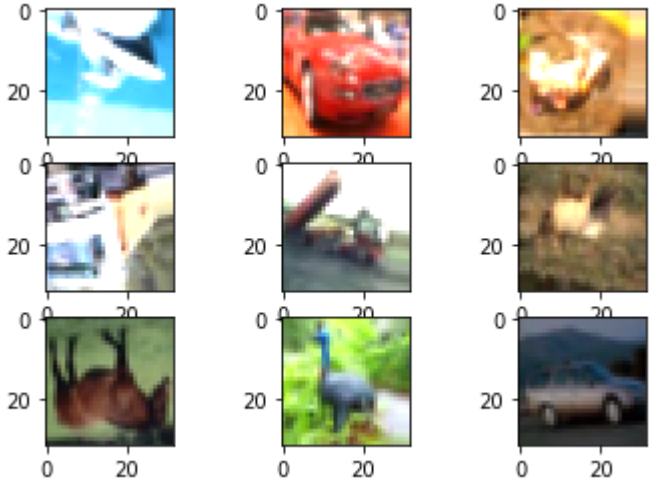
```
CPU times: user 178 µs, sys: 35 µs, total: 213 µs
Wall time: 218 µs
```

```

for X_batch, y_batch in datagen.flow(X_train[:9], y_train[:9], batch_size=9):
    for i in range(0, 9):
        plt.subplot(330 + 1 + i)

        plt.imshow(X_batch[i].astype('uint8'), cmap=plt.get_cmap('prism'))
    plt.show()
    break
```

```
/usr/local/lib/python3.6/dist-packages/keras_preprocessing/image/image_data_generator
  warnings.warn('This ImageDataGenerator specifies '
/usr/local/lib/python3.6/dist-packages/keras_preprocessing/image/image_data_generator
  warnings.warn('This ImageDataGenerator specifies '
```



```
def dense_net2(xtrain,xtest, optim = Adam(),k_size=(3,3), b_size = batch_size, epoch =
          print('b_size:{} epochs:{} '.format(b_size,epoch))
          input = layers.Input(shape=(img_height, img_width, channel,))
          First_Conv2D = layers.Conv2D(num_filter, (3,3), use_bias=False ,padding='same

          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)


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

          reduce_lr = ReduceLROnPlateau(monitor = 'val_loss', factor = 0.1, patience =


          def decay_fn(epoch, lr):
              if epoch < 50:
                  return 0.001
              elif epoch >= 50 and epoch < 75:
                  return 0.0001
              else:
                  return 0.00001

          lr_scheduler = LearningRateScheduler(decay_fn)

          csv_logger = CSVLogger('training.log')

```

```
36
37
38
39          checkpoint = ModelCheckpoint('gdrive/My Drive/cnnoncifar/models/model-{epoch:
40                                       verbose=1, monitor='val_acc',save_best_only=True
41
42
43
44
45          model.compile(loss='categorical_crossentropy',
46                        optimizer=Adam(),
47                        metrics=['accuracy'])
48
49          print(model.summary())
50          model.fit_generator(
51            datagen.flow(xtrain, y_train, batch_size=b_size),
52            steps_per_epoch=(len(xtrain)/batch_size)*5,
53            epochs=epoch,
54            verbose = 1,
55            validation_data=(xtest, y_test),callbacks=[checkpoint])
56
57
58
59          score = model.evaluate(xtest, y_test, verbose=1)
60          print('Test loss:', score[0])
61          print('Test accuracy:', score[1])
62
63          return model
64
65
```

```
1 compression = 1.041
2 l = 9
3 num_filter = 24
```

```
1 model = dense_net2(X_train,X_test,epoch=150)
```

```
def decay_fn(epoch, lr):
    if epoch < 50:
        return 0.001
    elif epoch >= 50 and epoch < 75:
        return 0.0001
    else:
        return 0.00001

lr_scheduler = LearningRateScheduler(decay_fn)

csv_logger = CSVLogger('training.log')




checkpoint = ModelCheckpoint('gdrive/My Drive/cnnoncifar/models/model-{epoch:03d}-{acc:
                                        verbose=1, monitor='val_acc',save_best_only=True

```

```
model.load_weights('gdrive/My Drive/cnnoncifar/models/model-033-0.893153-0.879100.h5')


model.compile(loss='categorical_crossentropy',
              optimizer=Adam(),
              metrics=['accuracy'])

print(model.summary())
model.fit_generator(
    datagen.flow(X_train, y_train, batch_size=batch_size),
    steps_per_epoch=(len(X_train)/batch_size)*5,

    epochs=150, verbose = 1,initial_epoch = 32,
    validation_data=(X_test, y_test),
    callbacks=[checkpoint])
```

```
def decay_fn(epoch, lr):
    if epoch < 50:
        return 0.001
    elif epoch >= 50 and epoch < 75:
        return 0.0001
    else:
        return 0.00001

lr_scheduler = LearningRateScheduler(decay_fn)

csv_logger = CSVLogger('training.log')



checkpoint = ModelCheckpoint('gdrive/My Drive/cnnoncifar/models/model-{epoch:03d}-{acc:
                                       verbose=1, monitor='val_acc',save_best_only=True


model.load_weights('gdrive/My Drive/cnnoncifar/models/model-060-0.915397-0.901300.h5')


model.compile(loss='categorical_crossentropy',
```

```
              optimizer=Adam(),
              metrics=['accuracy'])

print(model.summary())
model.fit_generator(
    datagen.flow(X_train, y_train, batch_size=batch_size),
    steps_per_epoch=(len(X_train)/batch_size)*5,

    epochs=150, verbose = 1,initial_epoch = 61,
    validation_data=(X_test, y_test),
    callbacks=[checkpoint])
```

```
Model: "model_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_2 (InputLayer)            [(None, 32, 32, 3)]  0
__________________________________________________________________________________________________
conv2d_165 (Conv2D)             (None, 32, 32, 24)   648         input_2[0][0]
__________________________________________________________________________________________________
batch_normalization_164 (BatchN (None, 32, 32, 24)   96          conv2d_165[0][0]
__________________________________________________________________________________________________
activation_165 (Activation)     (None, 32, 32, 24)   0           batch_normalization_
__________________________________________________________________________________________________
conv2d_166 (Conv2D)             (None, 32, 32, 24)   5184        activation_165[0][0]
__________________________________________________________________________________________________
dropout_163 (Dropout)           (None, 32, 32, 24)   0           conv2d_166[0][0]
__________________________________________________________________________________________________
concatenate_160 (Concatenate)   (None, 32, 32, 48)   0           conv2d_165[0][0]
                                                                 dropout_163[0][0]
__________________________________________________________________________________________________
batch_normalization_165 (BatchN (None, 32, 32, 48)   192         concatenate_160[0][0
__________________________________________________________________________________________________
activation_166 (Activation)     (None, 32, 32, 48)   0           batch_normalization_
__________________________________________________________________________________________________
conv2d_167 (Conv2D)             (None, 32, 32, 24)   10368       activation_166[0][0]
__________________________________________________________________________________________________
dropout_164 (Dropout)           (None, 32, 32, 24)   0           conv2d_167[0][0]
__________________________________________________________________________________________________
concatenate_161 (Concatenate)   (None, 32, 32, 72)   0           concatenate_160[0][0
                                                                 dropout_164[0][0]
__________________________________________________________________________________________________
batch_normalization_166 (BatchN (None, 32, 32, 72)   288         concatenate_161[0][0
__________________________________________________________________________________________________
activation_167 (Activation)     (None, 32, 32, 72)   0           batch_normalization_
__________________________________________________________________________________________________
conv2d_168 (Conv2D)             (None, 32, 32, 24)   15552       activation_167[0][0]
__________________________________________________________________________________________________
dropout_165 (Dropout)           (None, 32, 32, 24)   0           conv2d_168[0][0]
__________________________________________________________________________________________________
concatenate_162 (Concatenate)   (None, 32, 32, 96)   0           concatenate_161[0][0
                                                                 dropout_165[0][0]
__________________________________________________________________________________________________
batch_normalization_167 (BatchN (None, 32, 32, 96)   384         concatenate_162[0][0
__________________________________________________________________________________________________
activation_168 (Activation)     (None, 32, 32, 96)   0           batch_normalization_
__________________________________________________________________________________________________
conv2d_169 (Conv2D)             (None, 32, 32, 24)   20736       activation_168[0][0]
__________________________________________________________________________________________________
dropout_166 (Dropout)           (None, 32, 32, 24)   0           conv2d_169[0][0]
__________________________________________________________________________________________________
concatenate_163 (Concatenate)   (None, 32, 32, 120)  0           concatenate_162[0][0
                                                                 dropout_166[0][0]
__________________________________________________________________________________________________
batch_normalization_168 (BatchN (None, 32, 32, 120)  480         concatenate_163[0][0
__________________________________________________________________________________________________
activation_169 (Activation)     (None, 32, 32, 120)  0           batch_normalization_
__________________________________________________________________________________________________
conv2d_170 (Conv2D)             (None, 32, 32, 24)   25920       activation_169[0][0]
__________________________________________________________________________________________________
dropout_167 (Dropout)           (None, 32, 32, 24)   0           conv2d_170[0][0]
__________________________________________________________________________________________________
concatenate_164 (Concatenate)   (None, 32, 32, 144)  0           concatenate_163[0][0
```

```
                                                                 dropout_167[0][0]
__________________________________________________________________________________________________
batch_normalization_169 (BatchN (None, 32, 32, 144)  576         concatenate_164[0][0
__________________________________________________________________________________________________
activation_170 (Activation)     (None, 32, 32, 144)  0           batch_normalization_
__________________________________________________________________________________________________
conv2d_171 (Conv2D)             (None, 32, 32, 24)   31104       activation_170[0][0]
__________________________________________________________________________________________________
dropout_168 (Dropout)           (None, 32, 32, 24)   0           conv2d_171[0][0]
__________________________________________________________________________________________________
concatenate_165 (Concatenate)   (None, 32, 32, 168)  0           concatenate_164[0][0
                                                                 dropout_168[0][0]
__________________________________________________________________________________________________
batch_normalization_170 (BatchN (None, 32, 32, 168)  672         concatenate_165[0][0
__________________________________________________________________________________________________
activation_171 (Activation)     (None, 32, 32, 168)  0           batch_normalization_
__________________________________________________________________________________________________
conv2d_172 (Conv2D)             (None, 32, 32, 24)   36288       activation_171[0][0]
__________________________________________________________________________________________________
dropout_169 (Dropout)           (None, 32, 32, 24)   0           conv2d_172[0][0]
__________________________________________________________________________________________________
concatenate_166 (Concatenate)   (None, 32, 32, 192)  0           concatenate_165[0][0
                                                                 dropout_169[0][0]
__________________________________________________________________________________________________
batch_normalization_171 (BatchN (None, 32, 32, 192)  768         concatenate_166[0][0
__________________________________________________________________________________________________
activation_172 (Activation)     (None, 32, 32, 192)  0           batch_normalization_
__________________________________________________________________________________________________
conv2d_173 (Conv2D)             (None, 32, 32, 24)   41472       activation_172[0][0]
__________________________________________________________________________________________________
dropout_170 (Dropout)           (None, 32, 32, 24)   0           conv2d_173[0][0]
__________________________________________________________________________________________________
concatenate_167 (Concatenate)   (None, 32, 32, 216)  0           concatenate_166[0][0
                                                                 dropout_170[0][0]
__________________________________________________________________________________________________
batch_normalization_172 (BatchN (None, 32, 32, 216)  864         concatenate_167[0][0
__________________________________________________________________________________________________
activation_173 (Activation)     (None, 32, 32, 216)  0           batch_normalization_
__________________________________________________________________________________________________
conv2d_174 (Conv2D)             (None, 32, 32, 24)   46656       activation_173[0][0]
__________________________________________________________________________________________________
dropout_171 (Dropout)           (None, 32, 32, 24)   0           conv2d_174[0][0]
__________________________________________________________________________________________________
concatenate_168 (Concatenate)   (None, 32, 32, 240)  0           concatenate_167[0][0
                                                                 dropout_171[0][0]
__________________________________________________________________________________________________
batch_normalization_173 (BatchN (None, 32, 32, 240)  960         concatenate_168[0][0
__________________________________________________________________________________________________
activation_174 (Activation)     (None, 32, 32, 240)  0           batch_normalization_
__________________________________________________________________________________________________
conv2d_175 (Conv2D)             (None, 32, 32, 24)   5760        activation_174[0][0]
__________________________________________________________________________________________________
dropout_172 (Dropout)           (None, 32, 32, 24)   0           conv2d_175[0][0]
__________________________________________________________________________________________________
average_pooling2d_4 (AveragePoo (None, 16, 16, 24)   0           dropout_172[0][0]
__________________________________________________________________________________________________
batch_normalization_174 (BatchN (None, 16, 16, 24)   96          average_pooling2d_4[
__________________________________________________________________________________________________
activation_175 (Activation)     (None, 16, 16, 24)   0           batch_normalization_
__________________________________________________________________________________________________
conv2d_176 (Conv2D)             (None, 16, 16, 24)   5184        activation_175[0][0]
```

dropout_173 (Dropout)	(None, 16, 16, 24)	0	conv2d_176[0][0]
concatenate_169 (Concatenate)	(None, 16, 16, 48)	0	average_pooling2d_4[dropout_173[0][0]
batch_normalization_175 (BatchN	(None, 16, 16, 48)	192	concatenate_169[0][0
activation_176 (Activation)	(None, 16, 16, 48)	0	batch_normalization_
conv2d_177 (Conv2D)	(None, 16, 16, 24)	10368	activation_176[0][0]
dropout_174 (Dropout)	(None, 16, 16, 24)	0	conv2d_177[0][0]
concatenate_170 (Concatenate)	(None, 16, 16, 72)	0	concatenate_169[0][0 dropout_174[0][0]
batch_normalization_176 (BatchN	(None, 16, 16, 72)	288	concatenate_170[0][0
activation_177 (Activation)	(None, 16, 16, 72)	0	batch_normalization_
conv2d_178 (Conv2D)	(None, 16, 16, 24)	15552	activation_177[0][0]
dropout_175 (Dropout)	(None, 16, 16, 24)	0	conv2d_178[0][0]
concatenate_171 (Concatenate)	(None, 16, 16, 96)	0	concatenate_170[0][0 dropout_175[0][0]
batch_normalization_177 (BatchN	(None, 16, 16, 96)	384	concatenate_171[0][0
activation_178 (Activation)	(None, 16, 16, 96)	0	batch_normalization_
conv2d_179 (Conv2D)	(None, 16, 16, 24)	20736	activation_178[0][0]
dropout_176 (Dropout)	(None, 16, 16, 24)	0	conv2d_179[0][0]
concatenate_172 (Concatenate)	(None, 16, 16, 120)	0	concatenate_171[0][0 dropout_176[0][0]
batch_normalization_178 (BatchN	(None, 16, 16, 120)	480	concatenate_172[0][0
activation_179 (Activation)	(None, 16, 16, 120)	0	batch_normalization_
conv2d_180 (Conv2D)	(None, 16, 16, 24)	25920	activation_179[0][0]
dropout_177 (Dropout)	(None, 16, 16, 24)	0	conv2d_180[0][0]
concatenate_173 (Concatenate)	(None, 16, 16, 144)	0	concatenate_172[0][0 dropout_177[0][0]
batch_normalization_179 (BatchN	(None, 16, 16, 144)	576	concatenate_173[0][0
activation_180 (Activation)	(None, 16, 16, 144)	0	batch_normalization_
conv2d_181 (Conv2D)	(None, 16, 16, 24)	31104	activation_180[0][0]
dropout_178 (Dropout)	(None, 16, 16, 24)	0	conv2d_181[0][0]
concatenate_174 (Concatenate)	(None, 16, 16, 168)	0	concatenate_173[0][0 dropout_178[0][0]
batch_normalization_180 (BatchN	(None, 16, 16, 168)	672	concatenate_174[0][0

activation_181 (Activation)	(None, 16, 16, 168)	0	batch_normalization_
conv2d_182 (Conv2D)	(None, 16, 16, 24)	36288	activation_181[0][0]
dropout_179 (Dropout)	(None, 16, 16, 24)	0	conv2d_182[0][0]
concatenate_175 (Concatenate)	(None, 16, 16, 192)	0	concatenate_174[0][0 dropout_179[0][0]
batch_normalization_181 (BatchN	(None, 16, 16, 192)	768	concatenate_175[0][0
activation_182 (Activation)	(None, 16, 16, 192)	0	batch_normalization_
conv2d_183 (Conv2D)	(None, 16, 16, 24)	41472	activation_182[0][0]
dropout_180 (Dropout)	(None, 16, 16, 24)	0	conv2d_183[0][0]
concatenate_176 (Concatenate)	(None, 16, 16, 216)	0	concatenate_175[0][0 dropout_180[0][0]
batch_normalization_182 (BatchN	(None, 16, 16, 216)	864	concatenate_176[0][0
activation_183 (Activation)	(None, 16, 16, 216)	0	batch_normalization_
conv2d_184 (Conv2D)	(None, 16, 16, 24)	46656	activation_183[0][0]
dropout_181 (Dropout)	(None, 16, 16, 24)	0	conv2d_184[0][0]
concatenate_177 (Concatenate)	(None, 16, 16, 240)	0	concatenate_176[0][0 dropout_181[0][0]
batch_normalization_183 (BatchN	(None, 16, 16, 240)	960	concatenate_177[0][0
activation_184 (Activation)	(None, 16, 16, 240)	0	batch_normalization_
conv2d_185 (Conv2D)	(None, 16, 16, 24)	5760	activation_184[0][0]
dropout_182 (Dropout)	(None, 16, 16, 24)	0	conv2d_185[0][0]
average_pooling2d_5 (AveragePoo	(None, 8, 8, 24)	0	dropout_182[0][0]
batch_normalization_184 (BatchN	(None, 8, 8, 24)	96	average_pooling2d_5[
activation_185 (Activation)	(None, 8, 8, 24)	0	batch_normalization_
conv2d_186 (Conv2D)	(None, 8, 8, 24)	5184	activation_185[0][0]
dropout_183 (Dropout)	(None, 8, 8, 24)	0	conv2d_186[0][0]
concatenate_178 (Concatenate)	(None, 8, 8, 48)	0	average_pooling2d_5[dropout_183[0][0]
batch_normalization_185 (BatchN	(None, 8, 8, 48)	192	concatenate_178[0][0
activation_186 (Activation)	(None, 8, 8, 48)	0	batch_normalization_
conv2d_187 (Conv2D)	(None, 8, 8, 24)	10368	activation_186[0][0]
dropout_184 (Dropout)	(None, 8, 8, 24)	0	conv2d_187[0][0]
concatenate_179 (Concatenate)	(None, 8, 8, 72)	0	concatenate_178[0][0

[illegible]	[illegible]	[illegible]	dropout_184[0][0]
batch_normalization_186 (BatchN	(None, 8, 8, 72)	288	concatenate_179[0][0
activation_187 (Activation)	(None, 8, 8, 72)	0	batch_normalization_
conv2d_188 (Conv2D)	(None, 8, 8, 24)	15552	activation_187[0][0]
dropout_185 (Dropout)	(None, 8, 8, 24)	0	conv2d_188[0][0]
concatenate_180 (Concatenate)	(None, 8, 8, 96)	0	concatenate_179[0][0 dropout_185[0][0]
batch_normalization_187 (BatchN	(None, 8, 8, 96)	384	concatenate_180[0][0
activation_188 (Activation)	(None, 8, 8, 96)	0	batch_normalization_
conv2d_189 (Conv2D)	(None, 8, 8, 24)	20736	activation_188[0][0]
dropout_186 (Dropout)	(None, 8, 8, 24)	0	conv2d_189[0][0]
concatenate_181 (Concatenate)	(None, 8, 8, 120)	0	concatenate_180[0][0 dropout_186[0][0]
batch_normalization_188 (BatchN	(None, 8, 8, 120)	480	concatenate_181[0][0
activation_189 (Activation)	(None, 8, 8, 120)	0	batch_normalization_
conv2d_190 (Conv2D)	(None, 8, 8, 24)	25920	activation_189[0][0]
dropout_187 (Dropout)	(None, 8, 8, 24)	0	conv2d_190[0][0]
concatenate_182 (Concatenate)	(None, 8, 8, 144)	0	concatenate_181[0][0 dropout_187[0][0]
batch_normalization_189 (BatchN	(None, 8, 8, 144)	576	concatenate_182[0][0
activation_190 (Activation)	(None, 8, 8, 144)	0	batch_normalization_
conv2d_191 (Conv2D)	(None, 8, 8, 24)	31104	activation_190[0][0]
dropout_188 (Dropout)	(None, 8, 8, 24)	0	conv2d_191[0][0]
concatenate_183 (Concatenate)	(None, 8, 8, 168)	0	concatenate_182[0][0 dropout_188[0][0]
batch_normalization_190 (BatchN	(None, 8, 8, 168)	672	concatenate_183[0][0
activation_191 (Activation)	(None, 8, 8, 168)	0	batch_normalization_
conv2d_192 (Conv2D)	(None, 8, 8, 24)	36288	activation_191[0][0]
dropout_189 (Dropout)	(None, 8, 8, 24)	0	conv2d_192[0][0]
concatenate_184 (Concatenate)	(None, 8, 8, 192)	0	concatenate_183[0][0 dropout_189[0][0]
batch_normalization_191 (BatchN	(None, 8, 8, 192)	768	concatenate_184[0][0
activation_192 (Activation)	(None, 8, 8, 192)	0	batch_normalization_
[illegible]	[illegible]	[illegible]	[illegible]

conv2d_193 (Conv2D)	(None, 8, 8, 24)	41472	activation_192[0][0]
dropout_190 (Dropout)	(None, 8, 8, 24)	0	conv2d_193[0][0]
concatenate_185 (Concatenate)	(None, 8, 8, 216)	0	concatenate_184[0][0 dropout_190[0][0]
batch_normalization_192 (BatchN	(None, 8, 8, 216)	864	concatenate_185[0][0
activation_193 (Activation)	(None, 8, 8, 216)	0	batch_normalization_
conv2d_194 (Conv2D)	(None, 8, 8, 24)	46656	activation_193[0][0]
dropout_191 (Dropout)	(None, 8, 8, 24)	0	conv2d_194[0][0]
concatenate_186 (Concatenate)	(None, 8, 8, 240)	0	concatenate_185[0][0 dropout_191[0][0]
batch_normalization_193 (BatchN	(None, 8, 8, 240)	960	concatenate_186[0][0
activation_194 (Activation)	(None, 8, 8, 240)	0	batch_normalization_
conv2d_195 (Conv2D)	(None, 8, 8, 24)	5760	activation_194[0][0]
dropout_192 (Dropout)	(None, 8, 8, 24)	0	conv2d_195[0][0]
average_pooling2d_6 (AveragePoo	(None, 4, 4, 24)	0	dropout_192[0][0]
batch_normalization_194 (BatchN	(None, 4, 4, 24)	96	average_pooling2d_6[
activation_195 (Activation)	(None, 4, 4, 24)	0	batch_normalization_
conv2d_196 (Conv2D)	(None, 4, 4, 24)	5184	activation_195[0][0]
dropout_193 (Dropout)	(None, 4, 4, 24)	0	conv2d_196[0][0]
concatenate_187 (Concatenate)	(None, 4, 4, 48)	0	average_pooling2d_6[dropout_193[0][0]
batch_normalization_195 (BatchN	(None, 4, 4, 48)	192	concatenate_187[0][0
activation_196 (Activation)	(None, 4, 4, 48)	0	batch_normalization_
conv2d_197 (Conv2D)	(None, 4, 4, 24)	10368	activation_196[0][0]
dropout_194 (Dropout)	(None, 4, 4, 24)	0	conv2d_197[0][0]
concatenate_188 (Concatenate)	(None, 4, 4, 72)	0	concatenate_187[0][0 dropout_194[0][0]
batch_normalization_196 (BatchN	(None, 4, 4, 72)	288	concatenate_188[0][0
activation_197 (Activation)	(None, 4, 4, 72)	0	batch_normalization_
conv2d_198 (Conv2D)	(None, 4, 4, 24)	15552	activation_197[0][0]
dropout_195 (Dropout)	(None, 4, 4, 24)	0	conv2d_198[0][0]
concatenate_189 (Concatenate)	(None, 4, 4, 96)	0	concatenate_188[0][0 dropout_195[0][0]
batch_normalization_197 (BatchN	(None, 4, 4, 96)	384	concatenate_189[0][0

activation_198 (Activation)	(None, 4, 4, 96)	0	batch_normalization_
conv2d_199 (Conv2D)	(None, 4, 4, 24)	20736	activation_198[0][0]
dropout_196 (Dropout)	(None, 4, 4, 24)	0	conv2d_199[0][0]
concatenate_190 (Concatenate)	(None, 4, 4, 120)	0	concatenate_189[0][0 dropout_196[0][0]
batch_normalization_198 (BatchN	(None, 4, 4, 120)	480	concatenate_190[0][0
activation_199 (Activation)	(None, 4, 4, 120)	0	batch_normalization_
conv2d_200 (Conv2D)	(None, 4, 4, 24)	25920	activation_199[0][0]
dropout_197 (Dropout)	(None, 4, 4, 24)	0	conv2d_200[0][0]
concatenate_191 (Concatenate)	(None, 4, 4, 144)	0	concatenate_190[0][0 dropout_197[0][0]
batch_normalization_199 (BatchN	(None, 4, 4, 144)	576	concatenate_191[0][0
activation_200 (Activation)	(None, 4, 4, 144)	0	batch_normalization_
conv2d_201 (Conv2D)	(None, 4, 4, 24)	31104	activation_200[0][0]
dropout_198 (Dropout)	(None, 4, 4, 24)	0	conv2d_201[0][0]
concatenate_192 (Concatenate)	(None, 4, 4, 168)	0	concatenate_191[0][0 dropout_198[0][0]
batch_normalization_200 (BatchN	(None, 4, 4, 168)	672	concatenate_192[0][0
activation_201 (Activation)	(None, 4, 4, 168)	0	batch_normalization_
conv2d_202 (Conv2D)	(None, 4, 4, 24)	36288	activation_201[0][0]
dropout_199 (Dropout)	(None, 4, 4, 24)	0	conv2d_202[0][0]
concatenate_193 (Concatenate)	(None, 4, 4, 192)	0	concatenate_192[0][0 dropout_199[0][0]
batch_normalization_201 (BatchN	(None, 4, 4, 192)	768	concatenate_193[0][0
activation_202 (Activation)	(None, 4, 4, 192)	0	batch_normalization_
conv2d_203 (Conv2D)	(None, 4, 4, 24)	41472	activation_202[0][0]
dropout_200 (Dropout)	(None, 4, 4, 24)	0	conv2d_203[0][0]
concatenate_194 (Concatenate)	(None, 4, 4, 216)	0	concatenate_193[0][0 dropout_200[0][0]
batch_normalization_202 (BatchN	(None, 4, 4, 216)	864	concatenate_194[0][0
activation_203 (Activation)	(None, 4, 4, 216)	0	batch_normalization_
conv2d_204 (Conv2D)	(None, 4, 4, 24)	46656	activation_203[0][0]
dropout_201 (Dropout)	(None, 4, 4, 24)	0	conv2d_204[0][0]

```
concatenate_195 (Concatenate)   (None, 4, 4, 240)    0           concatenate_194[0][0
                                                                 dropout_201[0][0]
__________________________________________________________________________________________________
batch_normalization_203 (BatchN (None, 4, 4, 240)    960         concatenate_195[0][0
__________________________________________________________________________________________________
activation_204 (Activation)     (None, 4, 4, 240)    0           batch_normalization_
__________________________________________________________________________________________________
average_pooling2d_7 (AveragePoo (None, 2, 2, 240)    0           activation_204[0][0]
__________________________________________________________________________________________________
conv2d_205 (Conv2D)             (None, 2, 2, 10)     2400        average_pooling2d_7[
__________________________________________________________________________________________________
global_max_pooling2d_1 (GlobalM (None, 10)           0           conv2d_205[0][0]
__________________________________________________________________________________________________
activation_205 (Activation)     (None, 10)           0           global_max_pooling2d
==================================================================================================
Total params: 974,568
Trainable params: 964,008
Non-trainable params: 10,560
__________________________________________________________________________________________________
None
/usr/local/lib/python3.6/dist-packages/keras_preprocessing/image/image_data_generator
  warnings.warn('This ImageDataGenerator specifies '
/usr/local/lib/python3.6/dist-packages/keras_preprocessing/image/image_data_generator
  warnings.warn('This ImageDataGenerator specifies '
Epoch 62/150
/usr/local/lib/python3.6/dist-packages/keras_preprocessing/image/image_data_generator
  warnings.warn('This ImageDataGenerator specifies '
/usr/local/lib/python3.6/dist-packages/keras_preprocessing/image/image_data_generator
  warnings.warn('This ImageDataGenerator specifies '
/usr/local/lib/python3.6/dist-packages/keras_preprocessing/image/image_data_generator
  warnings.warn('This ImageDataGenerator specifies '
/usr/local/lib/python3.6/dist-packages/keras_preprocessing/image/image_data_generator
  warnings.warn('This ImageDataGenerator specifies '
/usr/local/lib/python3.6/dist-packages/keras_preprocessing/image/image_data_generator
  warnings.warn('This ImageDataGenerator specifies '
/usr/local/lib/python3.6/dist-packages/keras_preprocessing/image/image_data_generator
  warnings.warn('This ImageDataGenerator specifies '
1953/1953 [============================>.] - ETA: 0s - loss: 0.2368 - acc: 0.9162Epoc
10000/1953 [==============================================================================

Epoch 00062: val_acc improved from -inf to 0.90170, saving model to gdrive/My Drive/c
1954/1953 [==============================] - 963s 493ms/step - loss: 0.2368 - acc: 0.
Epoch 63/150
1953/1953 [============================>.] - ETA: 0s - loss: 0.2354 - acc: 0.9157Epoc
10000/1953 [==============================================================================

Epoch 00063: val_acc did not improve from 0.90170
1954/1953 [==============================] - 891s 456ms/step - loss: 0.2354 - acc: 0.
Epoch 64/150
1953/1953 [============================>.] - ETA: 0s - loss: 0.2332 - acc: 0.9175Epoc
10000/1953 [==============================================================================

Epoch 00064: val_acc did not improve from 0.90170
1954/1953 [==============================] - 892s 456ms/step - loss: 0.2332 - acc: 0.
Epoch 65/150
1953/1953 [============================>.] - ETA: 0s - loss: 0.2312 - acc: 0.9183Epoc
10000/1953 [==============================================================================

Epoch 00065: val_acc did not improve from 0.90170
1954/1953 [==============================] - 893s 457ms/step - loss: 0.2312 - acc: 0.
Epoch 66/150
```

```
1953/1953 [============================>.] - ETA: 0s - loss: 0.2289 - acc: 0.9191Epoc
10000/1953 [==============================================================================

Epoch 00066: val_acc did not improve from 0.90170
1954/1953 [==============================] - 882s 452ms/step - loss: 0.2289 - acc: 0.
Epoch 67/150
1953/1953 [============================>.] - ETA: 0s - loss: 0.2300 - acc: 0.9183Epoc
10000/1953 [==============================================================================

Epoch 00067: val_acc did not improve from 0.90170
1954/1953 [==============================] - 877s 449ms/step - loss: 0.2299 - acc: 0.
Epoch 68/150
1953/1953 [============================>.] - ETA: 0s - loss: 0.2286 - acc: 0.9182Epoc
10000/1953 [==============================================================================

Epoch 00068: val_acc did not improve from 0.90170
1954/1953 [==============================] - 876s 448ms/step - loss: 0.2286 - acc: 0.
Epoch 69/150
1953/1953 [============================>.] - ETA: 0s - loss: 0.2285 - acc: 0.9190Epoc
10000/1953 [==============================================================================

Epoch 00069: val_acc did not improve from 0.90170
1954/1953 [==============================] - 875s 448ms/step - loss: 0.2286 - acc: 0.
Epoch 70/150
1953/1953 [============================>.] - ETA: 0s - loss: 0.2242 - acc: 0.9204Epoc
10000/1953 [==============================================================================

Epoch 00070: val_acc did not improve from 0.90170
1954/1953 [==============================] - 877s 449ms/step - loss: 0.2242 - acc: 0.
Epoch 71/150
1953/1953 [============================>.] - ETA: 0s - loss: 0.2252 - acc: 0.9207Epoc
10000/1953 [==============================================================================

Epoch 00071: val_acc improved from 0.90170 to 0.90240, saving model to gdrive/My Driv
1954/1953 [==============================] - 880s 450ms/step - loss: 0.2253 - acc: 0.
Epoch 72/150
1953/1953 [============================>.] - ETA: 0s - loss: 0.2222 - acc: 0.9211Epoc
10000/1953 [==============================================================================

Epoch 00072: val_acc did not improve from 0.90240
1954/1953 [==============================] - 881s 451ms/step - loss: 0.2222 - acc: 0.
Epoch 73/150
1953/1953 [============================>.] - ETA: 0s - loss: 0.2210 - acc: 0.9212Epoc
10000/1953 [==============================================================================

Epoch 00073: val_acc did not improve from 0.90240
1954/1953 [==============================] - 886s 454ms/step - loss: 0.2211 - acc: 0.
Epoch 74/150
 654/1953 [=========>....................] - ETA: 9:47 - loss: 0.2212 - acc: 0.9207
```

```
model.load_weights('gdrive/My Drive/cnnoncifar/models/model-071-0.920683-0.902400.h5')
model.compile(loss='categorical_crossentropy',
                      optimizer=Adam(),
                      metrics=['accuracy'])

train_acc = model.evaluate(X_train,y_train)
val_acc   = model.evaluate(X_test,y_test)

```

```
50000/50000 [==============================] - 60s 1ms/sample - loss: 0.1094 - acc: 0
10000/10000 [==============================] - 11s 1ms/sample - loss: 0.3466 - acc: 0
```

```
print('The train accuracy is     : {}%'.format(96))
print('The test accuracy is      : {} i.e ~{}%'.format(90.24,91))
print('Number of parameters used : {}'.format(model.count_params()))
```

```
The train accuracy is     : 96%
The test accuracy is      : 90.24 i.e ~91%
Number of parameters used : 974568
```