CNN on CIFR 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 DropOut layers.
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.

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

```
from tensorflow.python.client import device_lib
device_lib.list_local_devices()
```

```
[name: "/device:CPU:0"
 device_type: "CPU"
 memory_limit: 268435456
 locality {
 }
 incarnation: 6447702113979293328, name: "/device:GPU:0"
 device_type: "GPU"
 memory_limit: 11345264640
 locality {
   bus_id: 1
```

```
    links {
    }
  }
  incarnation: 16374535067227115615
  physical_device_desc: "device: 0, name: Tesla K80, pci bus id: 0000:00:04.0, compute ca
```

```
import os
root_path = '/content/gdrive/MyDrive/cifar-10'


import warnings
warnings.filterwarnings("ignore")


# import keras
# from keras.datasets import cifar10
# from keras.models import Model, Sequential
# from keras.layers import Dense, Dropout, Flatten, Input, AveragePooling2D, merge, Activatio
# 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
from tensorflow.keras.optimizers import Adam


# 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 = 30
l = 6
num_filter = 64
compression = 0.5
dropout_rate = 0


# 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 [==============================] - 2s 0us/step
```

```
170508288/170498071 [==============================] - 2s 0us/step
```

```
X_train.shape
```

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

```
y_train.shape
```

```
(50000, 10)
```

```
X_test.shape
```

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

```
y_test.shape
```

```
(10000, 10)
```

Data Augmentation:

- Data augmentation involves making copies of the examples in the training dataset with small random modifications.
- This has a regularizing effect as it both expands the training dataset and allows the model to learn the same general features, although in a more generalized manner.

```
#https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-clas
from keras.preprocessing.image import ImageDataGenerator
# define data preparation
# Using image augmentation techniques
datagen = ImageDataGenerator(
        featurewise_center=False,  # set input mean to 0 over the dataset
        samplewise_center=False,  # set each sample mean to 0
        zca_whitening=False,  # apply ZCA whitening
        rotation_range=12,  # randomly rotate images in the range (degrees, 0 to 180)
        zoom_range = 0.15, # Randomly zoom image
        width_shift_range=0.15,  # randomly shift images horizontally (fraction of total widt
        height_shift_range=0.15,  # randomly shift images vertically (fraction of total heigh
        horizontal_flip=True,  # randomly flip images
        vertical_flip=False)  # randomly flip images
datagen.fit(X_train)
```

```
import matplotlib.pyplot as plt
import numpy as np
# configure batch size and retrieve one batch of images
for X_batch, y_batch in datagen.flow(X_train, y_train, batch_size=9):
  # create a grid of 3x3 images
```

```
# create a grid of 3x3 images
for i in range(0, 9):
  plt.subplot(330 + 1 + i)
  plt.imshow(X_batch[i].astype(np.uint8))
# show the plot
plt.show()
break
```

Define: Dense Block , Transition Block , Output Block.

```
# 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 ,paddin
        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 ,pad
    if dropout_rate>0:
         Conv2D_BottleNeck = layers.Dropout(dropout_rate)(Conv2D_BottleNeck)
```

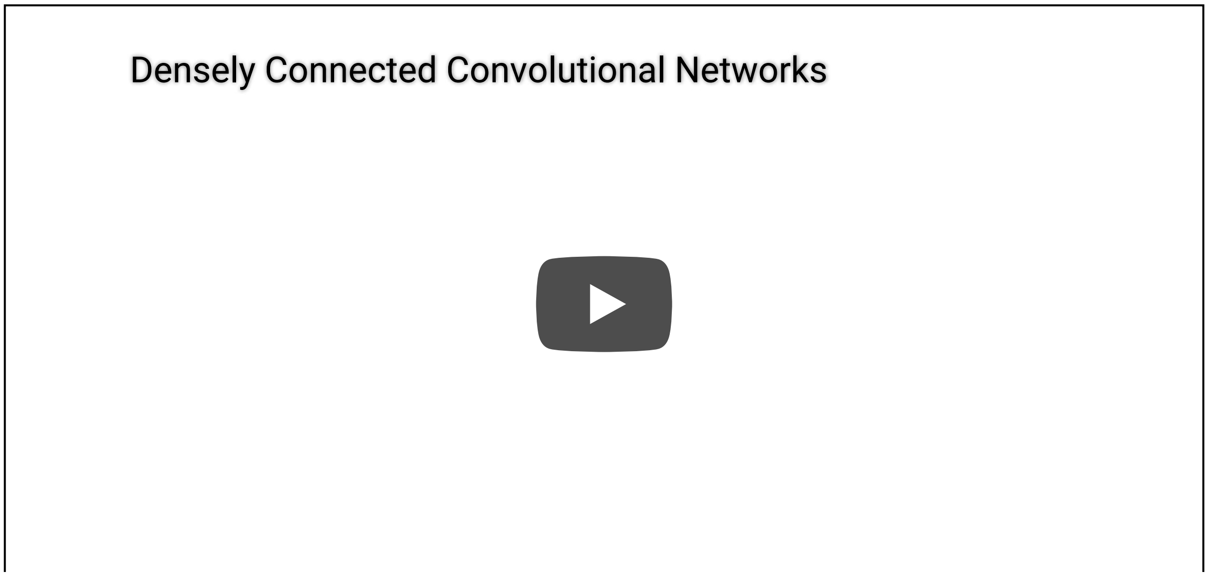
```
    avg = layers.AveragePooling2D(pool_size=(2,2))(Conv2D_BottleNeck)
    return avg

#output layer
def output_layer(input):
    global compression
    BatchNorm = layers.BatchNormalization()(input)
    relu = layers.Activation('relu')(BatchNorm)
    cv = layers.Conv2D(14, (1,1), use_bias=False ,padding='same')(relu)
    AvgPooling = layers.AveragePooling2D(pool_size=(2,2))(cv)
    flat = layers.Flatten()(AvgPooling)
    output = layers.Dense(num_classes, activation='softmax')(flat)
    return output
```

Define Model with some paramter.

- num_filter = 64
- l = 6
- Dropout_rate = 0
- Batch_size = 128

```
num_filter = 64
dropout_rate = 0
l = 6
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)
```



Here's the complete architecture of your model:

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

```
activation_2 (Activation)       (None, 32, 32, 128)  0           batch_normalization_
__________________________________________________________________________________________
conv2d_3 (Conv2D)               (None, 32, 32, 32)   36864       activation_2[0][0]
__________________________________________________________________________________________
concatenate_2 (Concatenate)     (None, 32, 32, 160)  0           concatenate_1[0][0]
                                                                 conv2d_3[0][0]
__________________________________________________________________________________________
batch_normalization_3 (BatchNor (None, 32, 32, 160)  640         concatenate_2[0][0]
__________________________________________________________________________________________
activation_3 (Activation)       (None, 32, 32, 160)  0           batch_normalization_
__________________________________________________________________________________________
conv2d_4 (Conv2D)               (None, 32, 32, 32)   46080       activation_3[0][0]
__________________________________________________________________________________________
concatenate_3 (Concatenate)     (None, 32, 32, 192)  0           concatenate_2[0][0]
                                                                 conv2d_4[0][0]
__________________________________________________________________________________________
batch_normalization_4 (BatchNor (None, 32, 32, 192)  768         concatenate_3[0][0]
__________________________________________________________________________________________
activation_4 (Activation)       (None, 32, 32, 192)  0           batch_normalization_
__________________________________________________________________________________________
conv2d_5 (Conv2D)               (None, 32, 32, 32)   55296       activation_4[0][0]
__________________________________________________________________________________________
concatenate_4 (Concatenate)     (None, 32, 32, 224)  0           concatenate_3[0][0]
                                                                 conv2d_5[0][0]
__________________________________________________________________________________________
batch_normalization_5 (BatchNor (None, 32, 32, 224)  896         concatenate_4[0][0]
__________________________________________________________________________________________
activation_5 (Activation)       (None, 32, 32, 224)  0           batch_normalization_
__________________________________________________________________________________________
conv2d_6 (Conv2D)               (None, 32, 32, 32)   64512       activation_5[0][0]
__________________________________________________________________________________________
concatenate_5 (Concatenate)     (None, 32, 32, 256)  0           concatenate_4[0][0]
                                                                 conv2d_6[0][0]
```

```
batch_normalization_6 (BatchNor (None, 32, 32, 256)  1024        concatenate_5[0][0]
activation_6 (Activation)       (None, 32, 32, 256)  0           batch_normalization_
conv2d_7 (Conv2D)               (None, 32, 32, 32)   8192        activation_6[0][0]
average_pooling2d (AveragePooli (None, 16, 16, 32)   0           conv2d_7[0][0]
batch_normalization_7 (BatchNor (None, 16, 16, 32)   128         average_pooling2d[0]
activation_7 (Activation)       (None, 16, 16, 32)   0           batch_normalization_
conv2d_8 (Conv2D)               (None, 16, 16, 32)   9216        activation_7[0][0]
concatenate_6 (Concatenate)     (None, 16, 16, 64)   0           average_pooling2d[0]
                                                                 conv2d_8[0][0]
batch_normalization_8 (BatchNor (None, 16, 16, 64)   256         concatenate_6[0][0]
activation_8 (Activation)       (None, 16, 16, 64)   0           batch_normalization_
conv2d_9 (Conv2D)               (None, 16, 16, 32)   18432       activation_8[0][0]
concatenate_7 (Concatenate)     (None, 16, 16, 96)   0           concatenate_6[0][0]
```

As we seen Total parameter is less than 1 millon which satisfy our assignment condition, So we can processed next operation.

```
print(f"Number of layers in model: {len(model.layers)}")
```

```
Number of layers in model: 116
```

```
import warnings
warnings.filterwarnings("ignore")
```

```
import os
checkpoint_path = "/content/gdrive/MyDrive/cifar-10/cp1-{epoch:04d}.ckpt"
checkpoint_dir = os.path.dirname(checkpoint_path)
```

```
cp_callback = tf.keras.callbacks.ModelCheckpoint(
    checkpoint_path, verbose=1, save_weights_only=True)
```

Compile the model

```
from tensorflow.keras.optimizers import SGD
```

```
# compile model
opt = SGD(lr=0.001, momentum=0.9)
model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
```

```
steps_per_epoch = X_train.shape[0]//batch_size
steps_per_epoch
```

```
390
```

Prepare Pixel Data

We know that the pixel values for each image in the dataset are unsigned integers in the range between no color and full color, or 0 and 255.

The prep_pixels() function below implement these behaviors and is provided with the pixel values for both the train and test datasets that will need to be scaled.

```
#https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-clas
# scale pixels
def prep_pixels(train, test):
  # convert from integers to floats
  train_norm = train.astype('float32')
  test_norm = test.astype('float32')
  # normalize to range 0-1
  train_norm = train_norm / 255.0
  test_norm = test_norm / 255.0
  # return normalized images
  return train_norm, test_norm
```

```
# prepare pixel data
X_train,X_test=prep_pixels(X_train,X_test)
```

train the model

```
#https://machinelearningmastery.com/check-point-deep-learning-models-keras/
from keras.callbacks import ModelCheckpoint

history = model.fit_generator(datagen.flow(X_train, y_train, batch_size=batch_size),steps_per
    epochs=epochs,
    verbose = 1,validation_steps = X_test.shape[0]//batch_size,
    validation_data=(X_test, y_test),callbacks=[cp_callback])
```

```
Epoch 16/30
390/390 [==============================] - 116s 298ms/step - loss: 0.6969 - accuracy:

Epoch 00016: saving model to /content/gdrive/MyDrive/cifar-10/cp1-0016.ckpt
```

```
Epoch 00016: saving model to /content/gdrive/MyDrive/cifar-10/cp1-0016.ckpt
Epoch 17/30
390/390 [==============================] - 116s 298ms/step - loss: 0.6763 - accuracy:

Epoch 00017: saving model to /content/gdrive/MyDrive/cifar-10/cp1-0017.ckpt
Epoch 18/30
390/390 [==============================] - 116s 298ms/step - loss: 0.6629 - accuracy:

Epoch 00018: saving model to /content/gdrive/MyDrive/cifar-10/cp1-0018.ckpt
Epoch 19/30
390/390 [==============================] - 116s 298ms/step - loss: 0.6492 - accuracy:

Epoch 00019: saving model to /content/gdrive/MyDrive/cifar-10/cp1-0019.ckpt
Epoch 20/30
390/390 [==============================] - 116s 298ms/step - loss: 0.6287 - accuracy:

Epoch 00020: saving model to /content/gdrive/MyDrive/cifar-10/cp1-0020.ckpt
Epoch 21/30
390/390 [==============================] - 116s 298ms/step - loss: 0.6181 - accuracy:

Epoch 00021: saving model to /content/gdrive/MyDrive/cifar-10/cp1-0021.ckpt
Epoch 22/30
390/390 [==============================] - 117s 299ms/step - loss: 0.5999 - accuracy:

Epoch 00022: saving model to /content/gdrive/MyDrive/cifar-10/cp1-0022.ckpt
Epoch 23/30
390/390 [==============================] - 117s 298ms/step - loss: 0.5858 - accuracy:

Epoch 00023: saving model to /content/gdrive/MyDrive/cifar-10/cp1-0023.ckpt
Epoch 24/30
390/390 [==============================] - 116s 298ms/step - loss: 0.5761 - accuracy:

Epoch 00024: saving model to /content/gdrive/MyDrive/cifar-10/cp1-0024.ckpt
Epoch 25/30
390/390 [==============================] - 116s 298ms/step - loss: 0.5619 - accuracy:

Epoch 00025: saving model to /content/gdrive/MyDrive/cifar-10/cp1-0025.ckpt
Epoch 26/30
390/390 [==============================] - 116s 298ms/step - loss: 0.5527 - accuracy:

Epoch 00026: saving model to /content/gdrive/MyDrive/cifar-10/cp1-0026.ckpt
Epoch 27/30
390/390 [==============================] - 116s 298ms/step - loss: 0.5379 - accuracy:

Epoch 00027: saving model to /content/gdrive/MyDrive/cifar-10/cp1-0027.ckpt
Epoch 28/30
390/390 [==============================] - 116s 298ms/step - loss: 0.5307 - accuracy:

Epoch 00028: saving model to /content/gdrive/MyDrive/cifar-10/cp1-0028.ckpt
Epoch 29/30
390/390 [==============================] - 116s 298ms/step - loss: 0.5268 - accuracy:

Epoch 00029: saving model to /content/gdrive/MyDrive/cifar-10/cp1-0029.ckpt
Epoch 30/30
390/390 [==============================] - 116s 298ms/step - loss: 0.5135 - accuracy:
```

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

```
313/313 [==============================] - 10s 30ms/step - loss: 0.8106 - accuracy: 0.75
Test loss: 0.8105748295783997
Test accuracy: 0.7529000043869019
```

```
model.load_weights('/content/gdrive/MyDrive/cifar-10/cp1-0030.ckpt')
```

```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f59717d5490>
```

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

```
Saved model to disk
```

As we seen above we get that we don't Get Good accuracy with This Parameter.
Now instead of using sgd we used Adam compiler with same parameter let's see wha
Adam compiler beacuse Adam is the best optimizers. If one wants to train the neu

```
#checkpoints after 30th epoch
import os
checkpoint_path = "/content/gdrive/My Drive/densenet_training/cp1-30-{epoch:04d}.ckpt"
checkpoint_dir = os.path.dirname(checkpoint_path)


cp_callback = tf.keras.callbacks.ModelCheckpoint(
    checkpoint_path, verbose=1, save_weights_only=True)


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


#https://machinelearningmastery.com/check-point-deep-learning-models-keras/
from keras.callbacks import ModelCheckpoint

history = model.fit_generator(datagen.flow(X_train, y_train, batch_size=batch_size),steps_per
    epochs=20,
```

```
verbose = 1,validation_steps = X_test.shape[0]//batch_size,
validation_data=(X_test, y_test),callbacks=[cp_callback])
```

```
390/390 [==============================] - 115s 295ms/step - loss: 0.5123 - accuracy:

Epoch 00006: saving model to /content/gdrive/My Drive/densenet_training/cp1-30-0006.c
Epoch 7/20
390/390 [==============================] - 115s 294ms/step - loss: 0.4873 - accuracy:

Epoch 00007: saving model to /content/gdrive/My Drive/densenet_training/cp1-30-0007.c
Epoch 8/20
390/390 [==============================] - 116s 296ms/step - loss: 0.4645 - accuracy:

Epoch 00008: saving model to /content/gdrive/My Drive/densenet_training/cp1-30-0008.c
Epoch 9/20
390/390 [==============================] - 115s 295ms/step - loss: 0.4395 - accuracy:

Epoch 00009: saving model to /content/gdrive/My Drive/densenet_training/cp1-30-0009.c
Epoch 10/20
390/390 [==============================] - 115s 296ms/step - loss: 0.4202 - accuracy:

Epoch 00010: saving model to /content/gdrive/My Drive/densenet_training/cp1-30-0010.c
Epoch 11/20
390/390 [==============================] - 115s 296ms/step - loss: 0.4035 - accuracy:

Epoch 00011: saving model to /content/gdrive/My Drive/densenet_training/cp1-30-0011.c
Epoch 12/20
390/390 [==============================] - 115s 296ms/step - loss: 0.3844 - accuracy:

Epoch 00012: saving model to /content/gdrive/My Drive/densenet_training/cp1-30-0012.c
Epoch 13/20
390/390 [==============================] - 115s 296ms/step - loss: 0.3708 - accuracy:

Epoch 00013: saving model to /content/gdrive/My Drive/densenet_training/cp1-30-0013.c
Epoch 14/20
390/390 [==============================] - 116s 296ms/step - loss: 0.3608 - accuracy:

Epoch 00014: saving model to /content/gdrive/My Drive/densenet_training/cp1-30-0014.c
Epoch 15/20
390/390 [==============================] - 115s 295ms/step - loss: 0.3534 - accuracy:

Epoch 00015: saving model to /content/gdrive/My Drive/densenet_training/cp1-30-0015.c
Epoch 16/20
390/390 [==============================] - 115s 295ms/step - loss: 0.3395 - accuracy:

Epoch 00016: saving model to /content/gdrive/My Drive/densenet_training/cp1-30-0016.c
Epoch 17/20
390/390 [==============================] - 115s 295ms/step - loss: 0.3293 - accuracy:

Epoch 00017: saving model to /content/gdrive/My Drive/densenet_training/cp1-30-0017.c
Epoch 18/20
390/390 [==============================] - 115s 295ms/step - loss: 0.3210 - accuracy:

Epoch 00018: saving model to /content/gdrive/My Drive/densenet_training/cp1-30-0018.c
Epoch 19/20
390/390 [==============================] - 115s 294ms/step - loss: 0.3089 - accuracy:
```

```
Epoch 00019: saving model to /content/gdrive/My Drive/densenet_training/cp1-30-0019.c
Epoch 20/20
390/390 [==============================] - 115s 295ms/step - loss: 0.3049 - accuracy:
```

```
model.load_weights('/content/gdrive/My Drive/densenet_training/cp1-30-0020.ckpt')
```

```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f99c3ebfed0>
```

Adam Give slightly Good output As compare to Sgd but we can't our expected outp

```
#checkpoints after 50th epoch
import os
checkpoint_path = "/content/gdrive/My Drive/densenet_training/cp1-50-{epoch:04d}.ckpt"
checkpoint_dir = os.path.dirname(checkpoint_path)


cp_callback = tf.keras.callbacks.ModelCheckpoint(
    checkpoint_path, verbose=1, save_weights_only=True)


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


#https://machinelearningmastery.com/check-point-deep-learning-models-keras/
from keras.callbacks import ModelCheckpoint

history = model.fit_generator(datagen.flow(X_train, y_train, batch_size=batch_size),steps_per
    epochs=20,
    verbose = 1,validation_steps = X_test.shape[0]//batch_size,
    validation_data=(X_test, y_test),callbacks=[cp_callback])
```

```
Epoch 1/20
390/390 [==============================] - 153s 301ms/step - loss: 0.2897 - accuracy:

Epoch 00001: saving model to /content/gdrive/My Drive/densenet_training/cp1-50-0001.c
Epoch 2/20
390/390 [==============================] - 113s 289ms/step - loss: 0.2866 - accuracy:

Epoch 00002: saving model to /content/gdrive/My Drive/densenet_training/cp1-50-0002.c
Epoch 3/20
390/390 [==============================] - 113s 289ms/step - loss: 0.2787 - accuracy:

Epoch 00003: saving model to /content/gdrive/My Drive/densenet_training/cp1-50-0003.c
```

```
Epoch 4/20
390/390 [==============================] - 113s 289ms/step - loss: 0.2761 - accuracy:

Epoch 00004: saving model to /content/gdrive/My Drive/densenet_training/cp1-50-0004.c
Epoch 5/20
390/390 [==============================] - 113s 289ms/step - loss: 0.2631 - accuracy:

Epoch 00005: saving model to /content/gdrive/My Drive/densenet_training/cp1-50-0005.c
Epoch 6/20
390/390 [==============================] - 113s 289ms/step - loss: 0.2617 - accuracy:

Epoch 00006: saving model to /content/gdrive/My Drive/densenet_training/cp1-50-0006.c
Epoch 7/20
390/390 [==============================] - 113s 289ms/step - loss: 0.2541 - accuracy:

Epoch 00007: saving model to /content/gdrive/My Drive/densenet_training/cp1-50-0007.c
Epoch 8/20
390/390 [==============================] - 113s 289ms/step - loss: 0.2427 - accuracy:

Epoch 00008: saving model to /content/gdrive/My Drive/densenet_training/cp1-50-0008.c
Epoch 9/20
390/390 [==============================] - 113s 289ms/step - loss: 0.2387 - accuracy:

Epoch 00009: saving model to /content/gdrive/My Drive/densenet_training/cp1-50-0009.c
Epoch 10/20
390/390 [==============================] - 113s 289ms/step - loss: 0.2349 - accuracy:

Epoch 00010: saving model to /content/gdrive/My Drive/densenet_training/cp1-50-0010.c
Epoch 11/20
390/390 [==============================] - 113s 290ms/step - loss: 0.2310 - accuracy:

Epoch 00011: saving model to /content/gdrive/My Drive/densenet_training/cp1-50-0011.c
Epoch 12/20
390/390 [==============================] - 113s 290ms/step - loss: 0.2265 - accuracy:

Epoch 00012: saving model to /content/gdrive/My Drive/densenet_training/cp1-50-0012.c
Epoch 13/20
390/390 [==============================] - 114s 291ms/step - loss: 0.2185 - accuracy:

Epoch 00013: saving model to /content/gdrive/My Drive/densenet_training/cp1-50-0013.c
Epoch 14/20
390/390 [==============================] - 113s 291ms/step - loss: 0.2143 - accuracy:

Epoch 00014: saving model to /content/gdrive/My Drive/densenet_training/cp1-50-0014.c
Epoch 15/20
390/390 [==============================] - 113s 290ms/step - loss: 0.2099 - accuracy:
```

```
model.load_weights('/content/gdrive/My Drive/densenet_training/cp1-50-0020.ckpt')
```

```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f99bf2cf090>
```

NOT GET OUR EXPECTED RESULT , WE HAVE TO GO AGAIN.

```
#checkpoints after 70th epoch
import os
checkpoint_path = "/content/gdrive/My Drive/densenet_training/cp1-70-{epoch:04d}.ckpt"
checkpoint_dir = os.path.dirname(checkpoint_path)


cp_callback = tf.keras.callbacks.ModelCheckpoint(
    checkpoint_path, verbose=1, save_weights_only=True)


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


#https://machinelearningmastery.com/check-point-deep-learning-models-keras/
from keras.callbacks import ModelCheckpoint

history = model.fit_generator(datagen.flow(X_train, y_train, batch_size=batch_size),steps_per
    epochs=20,
    verbose = 1,validation_steps = X_test.shape[0]//batch_size,
    validation_data=(X_test, y_test),callbacks=[cp_callback])
```

```
390/390 [==============================] - 113s 290ms/step - loss: 0.1707 - accuracy:

Epoch 00006: saving model to /content/gdrive/My Drive/densenet_training/cp1-70-0006.c
Epoch 7/20
390/390 [==============================] - 113s 290ms/step - loss: 0.1681 - accuracy:

Epoch 00007: saving model to /content/gdrive/My Drive/densenet_training/cp1-70-0007.c
Epoch 8/20
390/390 [==============================] - 113s 289ms/step - loss: 0.1637 - accuracy:

Epoch 00008: saving model to /content/gdrive/My Drive/densenet_training/cp1-70-0008.c
Epoch 9/20
390/390 [==============================] - 113s 290ms/step - loss: 0.1643 - accuracy:

Epoch 00009: saving model to /content/gdrive/My Drive/densenet_training/cp1-70-0009.c
Epoch 10/20
390/390 [==============================] - 113s 291ms/step - loss: 0.1637 - accuracy:

Epoch 00010: saving model to /content/gdrive/My Drive/densenet_training/cp1-70-0010.c
Epoch 11/20
390/390 [==============================] - 113s 290ms/step - loss: 0.1546 - accuracy:

Epoch 00011: saving model to /content/gdrive/My Drive/densenet_training/cp1-70-0011.c
Epoch 12/20
390/390 [==============================] - 113s 290ms/step - loss: 0.1520 - accuracy:

Epoch 00012: saving model to /content/gdrive/My Drive/densenet_training/cp1-70-0012.c
Epoch 13/20
390/390 [==============================] - 113s 290ms/step - loss: 0.1547 - accuracy:
```

```
Epoch 00013: saving model to /content/gdrive/My Drive/densenet_training/cp1-70-0013.c
Epoch 14/20
390/390 [==============================] - 113s 290ms/step - loss: 0.1491 - accuracy:

Epoch 00014: saving model to /content/gdrive/My Drive/densenet_training/cp1-70-0014.c
Epoch 15/20
390/390 [==============================] - 113s 289ms/step - loss: 0.1490 - accuracy:

Epoch 00015: saving model to /content/gdrive/My Drive/densenet_training/cp1-70-0015.c
Epoch 16/20
390/390 [==============================] - 113s 289ms/step - loss: 0.1461 - accuracy:

Epoch 00016: saving model to /content/gdrive/My Drive/densenet_training/cp1-70-0016.c
Epoch 17/20
390/390 [==============================] - 113s 289ms/step - loss: 0.1460 - accuracy:

Epoch 00017: saving model to /content/gdrive/My Drive/densenet_training/cp1-70-0017.c
Epoch 18/20
390/390 [==============================] - 113s 289ms/step - loss: 0.1402 - accuracy:

Epoch 00018: saving model to /content/gdrive/My Drive/densenet_training/cp1-70-0018.c
Epoch 19/20
390/390 [==============================] - 113s 289ms/step - loss: 0.1398 - accuracy:

Epoch 00019: saving model to /content/gdrive/My Drive/densenet_training/cp1-70-0019.c
Epoch 20/20
390/390 [==============================] - 113s 290ms/step - loss: 0.1358 - accuracy:

Epoch 00020: saving model to /content/gdrive/My Drive/densenet_training/cp1-70-0020.c
```

```
model.load_weights('/content/gdrive/My Drive/densenet_training/cp1-70-0014.ckpt')
```

```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f99bfe676d0>
```

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

```
313/313 [==============================] - 10s 30ms/step - loss: 0.3428 - accuracy: 0.9(
Test loss: 0.3428269028663635
Test accuracy: 0.8999999761581421
```

Here we get our output at 84 epoch.

We Run it again and see it give even good result .

```
model.load_weights('/content/gdrive/My Drive/densenet_training/cp1-70-0020.ckpt')
```

```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f99bfe0c750>
```

```
#checkpoints after 90 epoch
import os
checkpoint_path = "/content/gdrive/My Drive/densenet_training/cp1-90-{epoch:04d}.ckpt"
checkpoint_dir = os.path.dirname(checkpoint_path)


cp_callback = tf.keras.callbacks.ModelCheckpoint(
    checkpoint_path, verbose=1, save_weights_only=True)


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


#https://machinelearningmastery.com/check-point-deep-learning-models-keras/
from keras.callbacks import ModelCheckpoint

history = model.fit_generator(datagen.flow(X_train, y_train, batch_size=batch_size),steps_per
    epochs=20,
    verbose = 1,validation_steps = X_test.shape[0]//batch_size,
    validation_data=(X_test, y_test),callbacks=[cp_callback])
```

```
390/390 [==============================] - 113s 289ms/step - loss: 0.1281 - accuracy:

Epoch 00006: saving model to /content/gdrive/My Drive/densenet_training/cp1-90-0006.c
Epoch 7/20
390/390 [==============================] - 116s 297ms/step - loss: 0.1239 - accuracy:


Epoch 00007: saving model to /content/gdrive/My Drive/densenet_training/cp1-90-0007.c
Epoch 8/20
390/390 [==============================] - 112s 287ms/step - loss: 0.1227 - accuracy:

Epoch 00008: saving model to /content/gdrive/My Drive/densenet_training/cp1-90-0008.c
Epoch 9/20
390/390 [==============================] - 113s 288ms/step - loss: 0.1242 - accuracy:

Epoch 00009: saving model to /content/gdrive/My Drive/densenet_training/cp1-90-0009.c
Epoch 10/20
390/390 [==============================] - 113s 288ms/step - loss: 0.1185 - accuracy:

Epoch 00010: saving model to /content/gdrive/My Drive/densenet_training/cp1-90-0010.c
Epoch 11/20
390/390 [==============================] - 113s 289ms/step - loss: 0.1182 - accuracy:

Epoch 00011: saving model to /content/gdrive/My Drive/densenet_training/cp1-90-0011.c
Epoch 12/20
390/390 [==============================] - 113s 288ms/step - loss: 0.1167 - accuracy:

Epoch 00012: saving model to /content/gdrive/My Drive/densenet_training/cp1-90-0012.c
```

```
Epoch 13/20
390/390 [==============================] - 113s 289ms/step - loss: 0.1132 - accuracy:

Epoch 00013: saving model to /content/gdrive/My Drive/densenet_training/cp1-90-0013.c
Epoch 14/20
390/390 [==============================] - 113s 289ms/step - loss: 0.1140 - accuracy:

Epoch 00014: saving model to /content/gdrive/My Drive/densenet_training/cp1-90-0014.c
Epoch 15/20
390/390 [==============================] - 113s 289ms/step - loss: 0.1085 - accuracy:

Epoch 00015: saving model to /content/gdrive/My Drive/densenet_training/cp1-90-0015.c
Epoch 16/20
390/390 [==============================] - 113s 289ms/step - loss: 0.1138 - accuracy:

Epoch 00016: saving model to /content/gdrive/My Drive/densenet_training/cp1-90-0016.c
Epoch 17/20
390/390 [==============================] - 113s 289ms/step - loss: 0.1103 - accuracy:

Epoch 00017: saving model to /content/gdrive/My Drive/densenet_training/cp1-90-0017.c
Epoch 18/20
390/390 [==============================] - 113s 289ms/step - loss: 0.1072 - accuracy:

Epoch 00018: saving model to /content/gdrive/My Drive/densenet_training/cp1-90-0018.c
Epoch 19/20
390/390 [==============================] - 113s 289ms/step - loss: 0.1086 - accuracy:

Epoch 00019: saving model to /content/gdrive/My Drive/densenet_training/cp1-90-0019.c
Epoch 20/20
390/390 [==============================] - 113s 289ms/step - loss: 0.1081 - accuracy:
```

```
model.load_weights('/content/gdrive/My Drive/densenet_training/cp1-90-0011.ckpt')
```

```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f99c1400bd0>
```

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

```
313/313 [==============================] - 9s 29ms/step - loss: 0.3680 - accuracy: 0.901
Test loss: 0.36804988980293274
Test accuracy: 0.9017000198364258
```

```
model.load_weights('/content/gdrive/My Drive/densenet_training/cp1-90-0020.ckpt')
```

```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f59f3d5db10>
```

Now instead of Adam we used Sgd and see how it give result.

```
#checkpoints after 90 epoch
import os
checkpoint_path = "/content/gdrive/My Drive/densenet_training/cp1-110-{epoch:04d}.ckpt"
checkpoint_dir = os.path.dirname(checkpoint_path)


cp_callback = tf.keras.callbacks.ModelCheckpoint(
    checkpoint_path, verbose=1, save_weights_only=True)


# determine Loss function and Optimizer
from tensorflow.keras.optimizers import SGD
model.compile(loss='categorical_crossentropy',
              optimizer=SGD(lr=0.1, decay=1e-4, momentum=0.9, nesterov=True),
              metrics=['accuracy'])


#https://machinelearningmastery.com/check-point-deep-learning-models-keras/
from keras.callbacks import ModelCheckpoint

history = model.fit_generator(datagen.flow(X_train, y_train, batch_size=batch_size),steps_per
    epochs=20,
    verbose = 1,validation_steps = X_test.shape[0]//batch_size,
    validation_data=(X_test, y_test),callbacks=[cp_callback])
```

```
390/390 [==============================] - 113s 290ms/step - loss: 0.0814 - accuracy:

Epoch 00003: saving model to /content/gdrive/My Drive/densenet_training/cp1-110-0003.
Epoch 4/20
390/390 [==============================] - 113s 289ms/step - loss: 0.0757 - accuracy:

Epoch 00004: saving model to /content/gdrive/My Drive/densenet_training/cp1-110-0004.
Epoch 5/20
390/390 [==============================] - 113s 290ms/step - loss: 0.0737 - accuracy:

Epoch 00005: saving model to /content/gdrive/My Drive/densenet_training/cp1-110-0005.
Epoch 6/20
390/390 [==============================] - 113s 289ms/step - loss: 0.0684 - accuracy:

Epoch 00006: saving model to /content/gdrive/My Drive/densenet_training/cp1-110-0006.
Epoch 7/20
390/390 [==============================] - 113s 289ms/step - loss: 0.0688 - accuracy:


Epoch 00007: saving model to /content/gdrive/My Drive/densenet_training/cp1-110-0007.
Epoch 8/20
390/390 [==============================] - 113s 289ms/step - loss: 0.0685 - accuracy:

Epoch 00008: saving model to /content/gdrive/My Drive/densenet_training/cp1-110-0008.
Epoch 9/20
390/390 [==============================] - 113s 289ms/step - loss: 0.0681 - accuracy:

Epoch 00009: saving model to /content/gdrive/My Drive/densenet_training/cp1-110-0009.
```

```
Epoch 10/20
390/390 [==============================] - 113s 289ms/step - loss: 0.0669 - accuracy:

Epoch 00010: saving model to /content/gdrive/My Drive/densenet_training/cp1-110-0010.
Epoch 11/20
390/390 [==============================] - 113s 290ms/step - loss: 0.0661 - accuracy:

Epoch 00011: saving model to /content/gdrive/My Drive/densenet_training/cp1-110-0011.
Epoch 12/20
390/390 [==============================] - 113s 291ms/step - loss: 0.0649 - accuracy:

Epoch 00012: saving model to /content/gdrive/My Drive/densenet_training/cp1-110-0012.
Epoch 13/20
390/390 [==============================] - 113s 291ms/step - loss: 0.0643 - accuracy:

Epoch 00013: saving model to /content/gdrive/My Drive/densenet_training/cp1-110-0013.
Epoch 14/20
390/390 [==============================] - 114s 291ms/step - loss: 0.0626 - accuracy:

Epoch 00014: saving model to /content/gdrive/My Drive/densenet_training/cp1-110-0014.
Epoch 15/20
390/390 [==============================] - 114s 291ms/step - loss: 0.0632 - accuracy:

Epoch 00015: saving model to /content/gdrive/My Drive/densenet_training/cp1-110-0015.
Epoch 16/20
390/390 [==============================] - 113s 290ms/step - loss: 0.0631 - accuracy:

Epoch 00016: saving model to /content/gdrive/My Drive/densenet_training/cp1-110-0016.
Epoch 17/20
390/390 [==============================] - 113s 291ms/step - loss: 0.0628 - accuracy:
```

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

```
313/313 [==============================] - 10s 31ms/step - loss: 0.2825 - accuracy: 0.92
Test loss: 0.28252753615379333
Test accuracy: 0.9244999885559082
```

we get our Best output at 110 epoch , which 92.45 accuracy.

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

```
Saved model to disk
```

Conclusion:

After using some image augmentation techniques like rotation,horizontal flipping, height and width shifting, as well as changing some hyperparameters, we were able to get a model with 92.45% test accuracy. Our deep learning model has less than 1 Million parameters, and we have reached the desired accuracy in less than 300 epochs.

Refernce:

- https://colab.research.google.com/drive/1NGQjke72AS93IOpNcnE9diQEg78-sU3C#scrollTo=c5Wksy8z5-rw.
- https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-classification/
- https://www.tensorflow.org/tutorials/images/cnn

0s completed at 2:08 PM