Cifar10 and Cifar100 classification using DenseNet

1. Cifar 10 Classification

```
import os
from skimage import io
import numpy as np
from keras import backend as K
from keras.datasets import cifar10
from keras.models import Model, Sequential
from keras.layers import Dense, Dropout, Flatten, Input, AveragePooling2D, merge, A
from keras.layers import Conv2D, MaxPooling2D, BatchNormalization, DepthwiseConv2D
from keras.layers import Concatenate
from keras.models import load_model
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ReduceLROnPlateau, EarlyStopping, LearningRateScheduler
from keras.callbacks import Callback
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
# Load CIFAR10 Data
from tensorflow.keras.datasets import cifar10, cifar100
from tensorflow.keras.optimizers import Adam
```

```
# Hyperparameters
batch_size = 64
num_classes = 10
epochs = 100
# no of layers in dense block
l = 12
num_filter = 35
compression = 1.0
dropout_rate = 0.1
```

```
from sklearn.utils.validation import check_consistent_length
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()
img_h, img_w, chnl = 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)

x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size = 0.1

x_train = x_train.astype('float32')
```

```
x_test = x_test.astype('float32')
x_val = x_val.astype('float32')
```

```
# Data augementation
datagen_train = ImageDataGenerator(
    rescale=1./ 255,
    height_shift_range=0.125,
    horizontal_flip=True,
    rotation_range=20,
    zoom_range=0.10,
    width_shift_range=0.125,
    fill_mode='nearest',
)

datagen_train.fit(x_train)

validation_datagen = ImageDataGenerator(
    rescale=1./ 255)
```

```
# Reference: https://www.kaggle.com/code/genesis16/densenet-93-accuracy
# Dense Block
def add_denseblock(input, num_filter = 12, dropout_rate = 0.05):
    global compression
    temp = input
    for _ in range(l):
        BatchNorm = BatchNormalization()(temp)
        relu = Activation('relu')(BatchNorm)
        Conv2D_3_3 = Conv2D(int(num_filter*compression), (3,3), use_bias=False ,pad
        if dropout_rate>0:
          Conv2D_3_3 = Dropout(dropout_rate)(Conv2D_3_3)
        concat = Concatenate(axis=-1)([temp,Conv2D_3_3])

        temp = concat

    return temp
```

```
def add_transition(input, num_filter = 12, dropout_rate = 0.2):
    global compression
    BatchNorm = BatchNormalization()(input)
    relu = Activation('relu')(BatchNorm)
    Conv2D_BottleNeck = Conv2D(int(num_filter*compression), (1,1), use_bias=False ,
    if dropout_rate>0:
      Conv2D_BottleNeck = Dropout(dropout_rate)(Conv2D_BottleNeck)
    avg = AveragePooling2D(pool_size=(2,2))(Conv2D_BottleNeck)

    return avg
```

```
def output_layer(input):
    global compression
```

```
    BatchNorm = BatchNormalization()(input)
    relu = Activation('relu')(BatchNorm)
    AvgPooling = AveragePooling2D(pool_size=(2,2))(relu)
    flat = Flatten()(AvgPooling)
    output = Dense(num_classes, activation='softmax')(flat)

    return output
```

```
input = Input(shape=(img_h, img_w, chnl,))
firstconv2d= Conv2D(num_filter, (3,3), use_bias=False ,padding='same')(input)

firstblock= add_denseblock(firstconv2d, num_filter, dropout_rate)
firsttransition = add_transition(firstblock, num_filter, dropout_rate)

secondblock = add_denseblock(firsttransition, num_filter, dropout_rate)
secondtransition = add_transition(secondblock, num_filter, dropout_rate)

thirdblock = add_denseblock(secondtransition, num_filter, dropout_rate)
thirdtransition = add_transition(thirdblock, num_filter, dropout_rate)

lastblock = add_denseblock(thirdtransition,  num_filter, dropout_rate)
output = output_layer(lastblock)
```

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

```
 dropout_47 (Dropout)           (None, 4, 4, 35)     0           ['conv2d_48[0

 concatenate_44 (Concatenate)   (None, 4, 4, 350)    0           ['concatenate
                                                                  'dropout_47[

 batch_normalization_48 (BatchN  (None, 4, 4, 350)   1400        ['concatenate
 ormalization)

 activation_48 (Activation)     (None, 4, 4, 350)    0           ['batch_norma

 conv2d_49 (Conv2D)             (None, 4, 4, 35)     110250      ['activation_

 dropout_48 (Dropout)           (None, 4, 4, 35)     0           ['conv2d_49[0

 concatenate_45 (Concatenate)   (None, 4, 4, 385)    0           ['concatenate
                                                                  'dropout_48[

 batch_normalization_49 (BatchN  (None, 4, 4, 385)   1540        ['concatenate
 ormalization)

 activation_49 (Activation)     (None, 4, 4, 385)    0           ['batch_norma

 conv2d_50 (Conv2D)             (None, 4, 4, 35)     121275      ['activation_

 dropout_49 (Dropout)           (None, 4, 4, 35)     0           ['conv2d_50[0

 concatenate_46 (Concatenate)   (None, 4, 4, 420)    0           ['concatenate
                                                                  'dropout_49[

 batch_normalization_50 (BatchN  (None, 4, 4, 420)   1680        ['concatenate
```

```
 batch_normalization_50 (BatchN  (None, 4, 4, 420)   1680        ['concatenate
 ormalization)

 activation_50 (Activation)     (None, 4, 4, 420)    0           ['batch_norma

 conv2d_51 (Conv2D)             (None, 4, 4, 35)     132300      ['activation_

 dropout_50 (Dropout)           (None, 4, 4, 35)     0           ['conv2d_51[0

 concatenate_47 (Concatenate)   (None, 4, 4, 455)    0           ['concatenate
                                                                  'dropout_50[

 batch_normalization_51 (BatchN  (None, 4, 4, 455)   1820        ['concatenate
 ormalization)

 activation_51 (Activation)     (None, 4, 4, 455)    0           ['batch_norma

 average_pooling2d_3 (AveragePo  (None, 2, 2, 455)   0           ['activation_
 oling2D)


 flatten (Flatten)              (None, 1820)         0           ['average_poo

 dense (Dense)                  (None, 10)           18210       ['flatten[0][

==============================================================================
Total params: 3,557,690
Trainable params: 3,532,210
Non-trainable params: 25,480
______________________________________________________________________________
```

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

```
reduce_lr = ReduceLROnPlateau(monitor = 'val_loss', factor = 0.1, patience = 5, min

early_stop = EarlyStopping(monitor = "val_loss", patience = 10)

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)
```

```
%%time
history = model.fit_generator(
    datagen_train.flow(x_train, y_train, batch_size=64),
    steps_per_epoch=len(x_train) // 64,
    epochs=epochs,
```

```
    verbose = 1,
    validation_data=validation_datagen.flow((x_val, y_val), batch_size=64),
    validation_steps = len(y_val)/64,
    callbacks = [lr_scheduler]
)
```

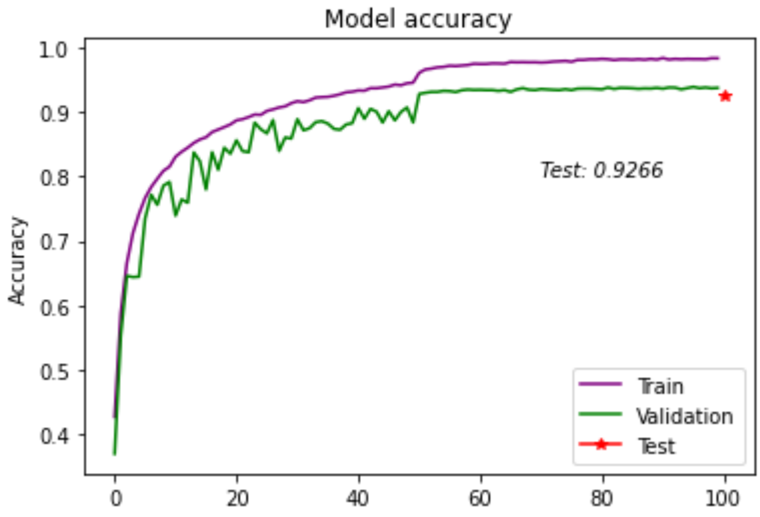
```
703/703 [==============================] - 100s 142ms/step - loss: 0.0642 - ac
703/703 [==============================] - 99s 141ms/step - loss: 0.0608 - acc
Epoch 74/100
703/703 [==============================] - 100s 142ms/step - loss: 0.0598 - ac
Epoch 75/100
703/703 [==============================] - 99s 141ms/step - loss: 0.0592 - acc
Epoch 76/100
703/703 [==============================] - 99s 141ms/step - loss: 0.0595 - acc
Epoch 77/100
703/703 [==============================] - 100s 142ms/step - loss: 0.0546 - ac
Epoch 78/100
703/703 [==============================] - 99s 141ms/step - loss: 0.0554 - acc
Epoch 79/100
703/703 [==============================] - 99s 141ms/step - loss: 0.0542 - acc
Epoch 80/100
703/703 [==============================] - 99s 141ms/step - loss: 0.0543 - acc
Epoch 81/100
703/703 [==============================] - 99s 141ms/step - loss: 0.0534 - acc
Epoch 82/100
703/703 [==============================] - 99s 141ms/step - loss: 0.0537 - acc
Epoch 83/100
703/703 [==============================] - 99s 141ms/step - loss: 0.0550 - acc
Epoch 84/100
703/703 [==============================] - 99s 141ms/step - loss: 0.0525 - acc
Epoch 85/100
703/703 [==============================] - 98s 139ms/step - loss: 0.0539 - acc
Epoch 86/100
703/703 [==============================] - 98s 139ms/step - loss: 0.0542 - acc
Epoch 87/100
703/703 [==============================] - 98s 139ms/step - loss: 0.0522 - acc
Epoch 88/100
703/703 [==============================] - 98s 139ms/step - loss: 0.0539 - acc
Epoch 89/100
703/703 [==============================] - 98s 139ms/step - loss: 0.0517 - acc
Epoch 90/100
703/703 [==============================] - 98s 139ms/step - loss: 0.0532 - acc
Epoch 91/100
703/703 [==============================] - 98s 139ms/step - loss: 0.0497 - acc
Epoch 92/100
703/703 [==============================] - 98s 139ms/step - loss: 0.0518 - acc
Epoch 93/100
703/703 [==============================] - 98s 140ms/step - loss: 0.0507 - acc
Epoch 94/100
703/703 [==============================] - 98s 139ms/step - loss: 0.0528 - acc
Epoch 95/100
703/703 [==============================] - 98s 139ms/step - loss: 0.0517 - acc
Epoch 96/100
703/703 [==============================] - 98s 139ms/step - loss: 0.0512 - acc
Epoch 97/100
703/703 [==============================] - 98s 139ms/step - loss: 0.0512 - acc
Epoch 98/100
703/703 [==============================] - 98s 140ms/step - loss: 0.0513 - acc
Epoch 99/100
703/703 [==============================] - 98s 140ms/step - loss: 0.0483 - acc
```

```
Epoch 100/100
703/703 [==============================] - 98s 139ms/step - loss: 0.0495 - acc
CPU times: user 3h 27min 40s, sys: 3min 40s, total: 3h 31min 20s
Wall time: 3h 27min 12s
```

```
x_test = x_test/255
test_loss, test_acc = model.evaluate(x_test, y_test)
print("Test accuracy:",test_acc)
```

```
313/313 [==============================] - 8s 26ms/step - loss: 0.3148 - accur
Test accuracy: 0.9265999794006348
```

```


import matplotlib.pyplot as plt
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(len(acc))

plt.plot(epochs, acc, 'purple', label='Train')
plt.plot(epochs, val_acc, 'g', label='Validation')
plt.plot(len(epochs), test_acc, 'r', marker="*", label='Test')
plt.text(70,0.8, "Test: 0.9266", style='italic')

plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend()
plt.figure()

plt.plot(epochs, loss, 'purple', label='Train')
plt.plot(epochs, val_loss, 'g', label='Validation')
plt.title('Loss')
plt.legend()


plt.show()
```



2. Cifar 100

```
# Hyperparameters
batch_size = 64
# change num_classes = 100
num_classes = 100
epochs = 100
# increase number of layers in dense block
l = 15
num_filter = 35
compression = 1
dropout_rate = 0.1
```

```
from sklearn.utils.validation import check_consistent_length
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar100.load_data()
img_h, img_w, chnl = 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)

x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size = 0.1

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_val = x_val.astype('float32')
```

```
# Data augementation
datagen_train = ImageDataGenerator(
    rescale=1./ 255,
    rotation_range=20,
    width_shift_range=0.125,
    height_shift_range=0.125,
    horizontal_flip=True,
```

```
    fill_mode='nearest',
    zoom_range=0.10
)

datagen_train.fit(x_train)

validation_datagen = ImageDataGenerator(
    rescale=1./ 255)
```

```
# Dense Block
def add_denseblock(input, num_filter = 12, dropout_rate = 0.05):
    global compression
    temp = input
    for _ in range(l):
        BatchNorm = BatchNormalization()(temp)
        relu = Activation('relu')(BatchNorm)
        Conv2D_3_3 = Conv2D(int(num_filter*compression), (3,3), use_bias=False ,pad
        if dropout_rate>0:
          Conv2D_3_3 = Dropout(dropout_rate)(Conv2D_3_3)
        concat = Concatenate(axis=-1)([temp,Conv2D_3_3])

        temp = concat

    return temp
```

```
def add_transition(input, num_filter = 12, dropout_rate = 0.2):
    global compression
    BatchNorm = BatchNormalization()(input)
    relu = Activation('relu')(BatchNorm)
    Conv2D_BottleNeck = Conv2D(int(num_filter*compression), (1,1), use_bias=False ,
    if dropout_rate>0:
      Conv2D_BottleNeck = Dropout(dropout_rate)(Conv2D_BottleNeck)
    avg = AveragePooling2D(pool_size=(2,2))(Conv2D_BottleNeck)

    return avg
```

```
def output_layer(input):
    global compression
    BatchNorm = BatchNormalization()(input)
    relu = Activation('relu')(BatchNorm)
    AvgPooling = AveragePooling2D(pool_size=(2,2))(relu)
    flat = Flatten()(AvgPooling)
    output = Dense(num_classes, activation='softmax')(flat)

    return output
```

```
input = Input(shape=(img_h, img_w, chnl,))
firstconv2d = Conv2D(num_filter, (3,3), use_bias=False ,padding='same')(input)

firstblock = add_denseblock(firstconv2d, num_filter, dropout_rate)
firsttransition = add_transition(firstblock, num_filter, dropout_rate)
```

```
secondblock = add_denseblock(firsttransition, num_filter, dropout_rate)
secondtransition = add_transition(secondblock, num_filter, dropout_rate)

thirdblock = add_denseblock(secondtransition, num_filter, dropout_rate)
thirdtransition = add_transition(thirdblock, num_filter, dropout_rate)

lastblock = add_denseblock(thirdtransition,  num_filter, dropout_rate)
output = output_layer(lastblock)
```

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

```
# determine Loss function and Optimizer
model.compile(loss='categorical_crossentropy',
              optimizer=Adam(),
              metrics=[tf.keras.metrics.TopKCategoricalAccuracy(k=5)])
```

```
reduce_lr = ReduceLROnPlateau(monitor = 'val_loss', factor = 0.1, patience = 5, min

early_stop = EarlyStopping(monitor = "val_loss", patience = 10)

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)
```

```
%%time
history = model.fit_generator(
    datagen_train.flow(x_train, y_train, batch_size=64),
    steps_per_epoch=len(x_train) // 64,
    epochs=epochs,
    verbose = 1,
    validation_data=validation_datagen.flow((x_val, y_val), batch_size=64),
    validation_steps = len(y_val)/64,
    callbacks = [lr_scheduler]
)
```

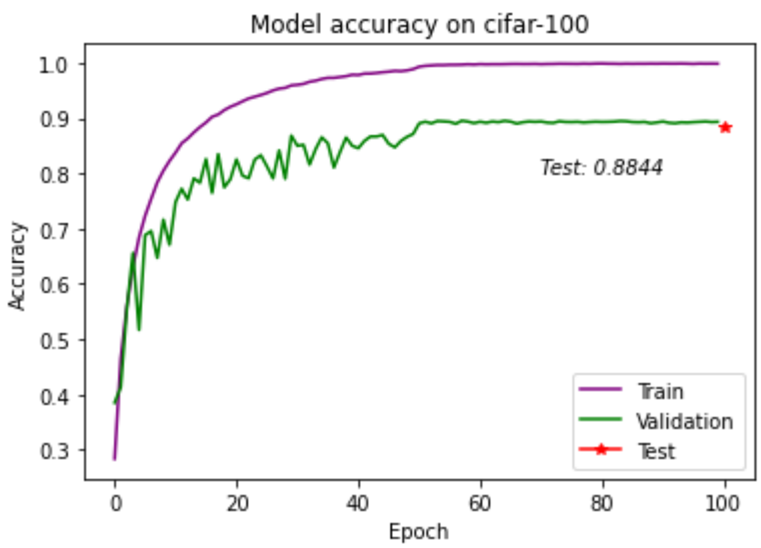
```
Epoch 73/100
703/703 [==============================] - 131s 187ms/step - loss: 0.1733 - tc
Epoch 74/100
703/703 [==============================] - 131s 186ms/step - loss: 0.1735 - tc
Epoch 75/100
703/703 [==============================] - 131s 186ms/step - loss: 0.1695 - tc
Epoch 76/100
703/703 [==============================] - 131s 187ms/step - loss: 0.1672 - tc
Epoch 77/100
```

```
Epoch 77/100
703/703 [==============================] - 131s 186ms/step - loss: 0.1578 - tc
Epoch 78/100
703/703 [==============================] - 131s 187ms/step - loss: 0.1589 - tc
Epoch 79/100
703/703 [==============================] - 131s 187ms/step - loss: 0.1597 - tc
Epoch 80/100
703/703 [==============================] - 131s 186ms/step - loss: 0.1567 - tc
Epoch 81/100
703/703 [==============================] - 133s 189ms/step - loss: 0.1561 - tc
Epoch 82/100
703/703 [==============================] - 131s 187ms/step - loss: 0.1551 - tc
Epoch 83/100
703/703 [==============================] - 131s 186ms/step - loss: 0.1547 - tc
Epoch 84/100
703/703 [==============================] - 131s 187ms/step - loss: 0.1575 - tc
Epoch 85/100
703/703 [==============================] - 131s 186ms/step - loss: 0.1573 - tc
Epoch 86/100
703/703 [==============================] - 131s 186ms/step - loss: 0.1564 - tc
Epoch 87/100
703/703 [==============================] - 131s 186ms/step - loss: 0.1578 - tc
Epoch 88/100
703/703 [==============================] - 131s 186ms/step - loss: 0.1528 - tc
Epoch 89/100
703/703 [==============================] - 131s 187ms/step - loss: 0.1520 - tc
Epoch 90/100
703/703 [==============================] - 131s 186ms/step - loss: 0.1530 - tc
Epoch 91/100
703/703 [==============================] - 131s 186ms/step - loss: 0.1483 - tc
Epoch 92/100
703/703 [==============================] - 131s 186ms/step - loss: 0.1495 - tc
Epoch 93/100
703/703 [==============================] - 131s 186ms/step - loss: 0.1529 - tc
Epoch 94/100
703/703 [==============================] - 131s 186ms/step - loss: 0.1467 - tc
Epoch 95/100
703/703 [==============================] - 131s 186ms/step - loss: 0.1512 - tc
Epoch 96/100
703/703 [==============================] - 131s 186ms/step - loss: 0.1518 - tc
Epoch 97/100
703/703 [==============================] - 131s 187ms/step - loss: 0.1492 - tc
Epoch 98/100
703/703 [==============================] - 131s 186ms/step - loss: 0.1461 - tc
Epoch 99/100
703/703 [==============================] - 131s 186ms/step - loss: 0.1501 - tc
Epoch 100/100
703/703 [==============================] - 131s 186ms/step - loss: 0.1508 - tc
CPU times: user 3h 37min 1s, sys: 3min 14s, total: 3h 40min 16s
Wall time: 3h 49min 15s
```

```
x_test = x_test/255
test_loss, test_acc = model.evaluate(x_test, y_test)
print("Test accuracy:",test_acc)
```

```
313/313 [==============================] - 12s 33ms/step - loss: 1.9677 - top_
Test accuracy: 0.8844000101089478
```

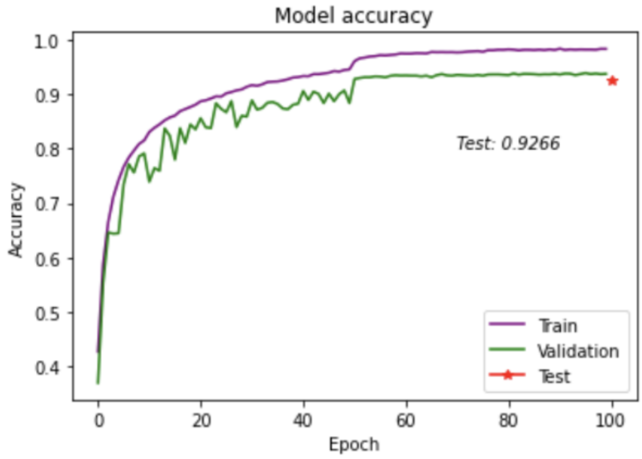
```
import matplotlib.pyplot as plt
acc = history.history['top_k_categorical_accuracy']
val_acc = history.history['val_top_k_categorical_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(len(acc))

plt.plot(epochs, acc, 'purple', label='Train')
plt.plot(epochs, val_acc, 'g', label='Validation')
plt.plot(len(epochs), test_acc, 'r', marker="*", label='Test')
plt.text(70,0.8, "Test: 0.8844", style='italic')

plt.title('Model accuracy on cifar-100')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend()
plt.figure()

plt.show()
```

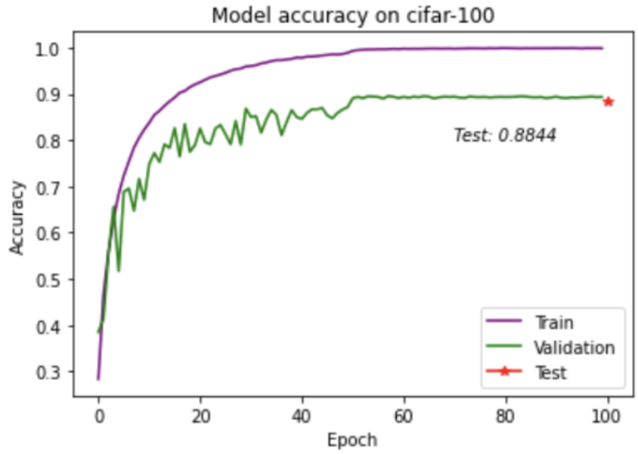



```
<Figure size 432x288 with 0 Axes>
```

Results

Acheived 92.66 % accuracy on Cifar-10 (left) and 88.44% accuracy on Cifar-100 (right)





I defined state-of-the-art accuracy as above 92. Accuracy of 92 was reached around 2014-2015, according to the Cifar10 benchmark (https://paperswithcode.com/sota/image-classification-on-cifar-10). To achieve this accuracy, I used DenseNet, which utilizes Dense Blocks. Dense Blocks are dense connections between each layer. All the layers are connected directly, and each layer obtains additional inputs from all preceding layers and passes on its feature maps to all subsequent layers. Concatenation is used to receive "collective knowledge" from all prior layers. "Bottleneck" layers are used with the Conv2D layer to reduce the model complexity and size.