Name : Pravalika Medasani

700# - 700744503

Github Link : <https://github.com/PravalikaMedasani/ICP_7>

# Import libraries

import numpy as np

import matplotlib.pyplot as plt import seaborn as sns

import tensorflow as tf

from tensorflow.keras.datasets import cifar100

from tensorflow.keras.optimizers import RMSprop from keras.preprocessing import image

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Dropout, BatchNormalization

%matplotlib inline

# Extract data and train and test dataset

#cifar100 = tf.keras.datasets.cifar100

(X\_train,Y\_train) , (X\_test,Y\_test) = cifar100.load\_data()

 Downloading data from <https://www.cs.toronto.edu/~kriz/cifar-100-python.tar.gz> 169001437/169001437 [==============================] - 4s 0us/step

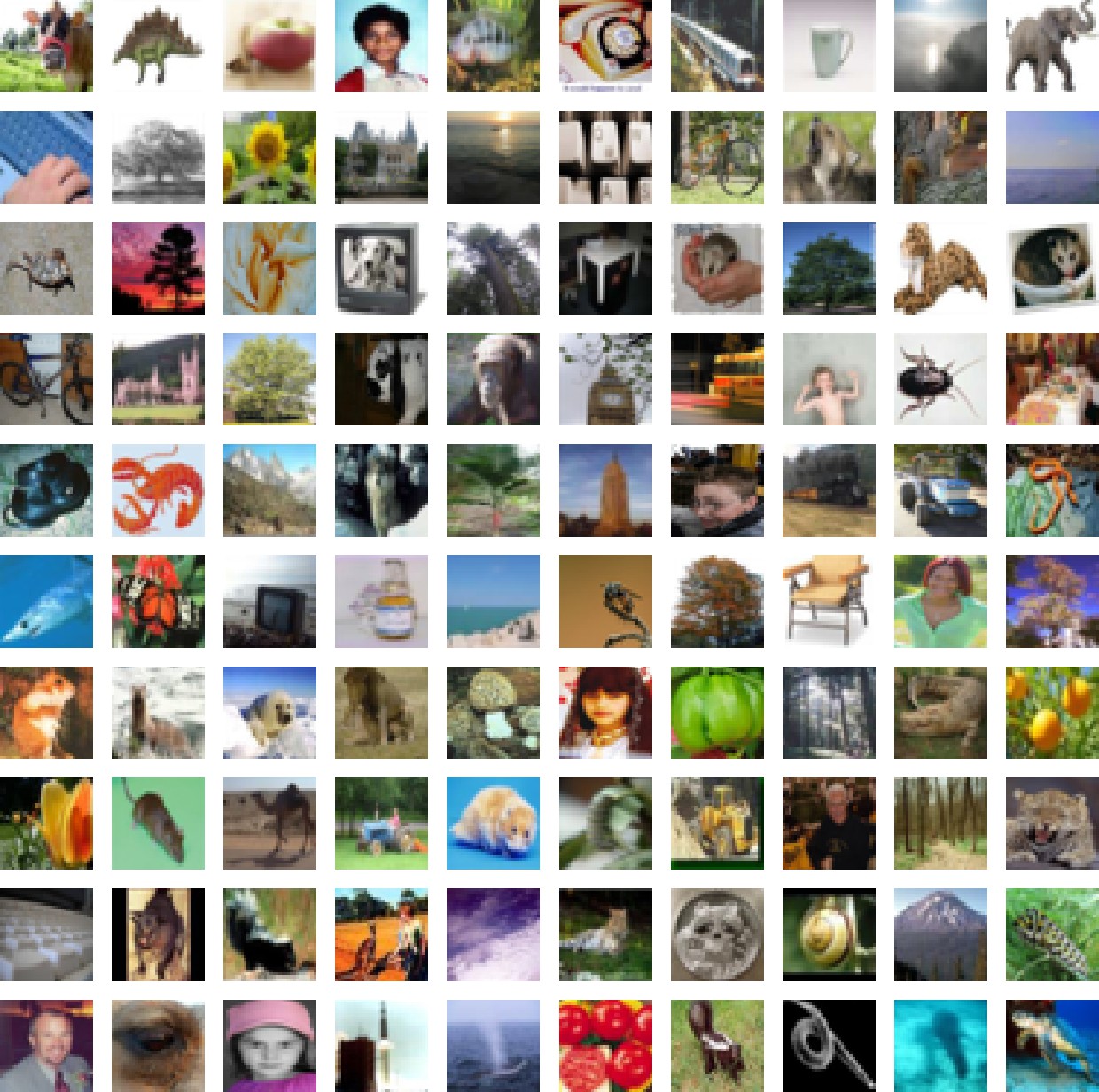
classes = ['apple', 'aquarium\_fish', 'baby', 'bear', 'beaver', 'bed', 'bee', 'beetle', 'bicycle', 'bottle', 'bowl', 'boy', 'bridge', 'bus

# Let's look into the dataset images

plt.figure(figsize = (16,16)) for i in range(100):

plt.subplot(10,10,1+i) plt.axis('off')

plt.imshow(X\_train[i], cmap = 'gray')



# Training , Validating and Splitting trained and tested data

from sklearn.model\_selection import train\_test\_split

x\_train, x\_val, y\_train, y\_val = train\_test\_split(X\_train,Y\_train,test\_size=0.2)

from tensorflow.keras.utils import to\_categorical

y\_train = to\_categorical(y\_train, num\_classes = 100) y\_val = to\_categorical(y\_val, num\_classes = 100)

print(x\_train.shape) print(y\_train.shape) print(x\_val.shape)

print(y\_val.shape) print(X\_test.shape) print(Y\_test.shape)

(40000, 32, 32, 3)

(40000, 100)

(10000, 32, 32, 3)

(10000, 100)

(10000, 32, 32, 3)

(10000, 1)

train\_datagen = ImageDataGenerator(

preprocessing\_function = tf.keras.applications.vgg19.preprocess\_input, rotation\_range=10,

zoom\_range = 0.1,

width\_shift\_range = 0.1,

height\_shift\_range = 0.1,

shear\_range = 0.1,

horizontal\_flip = True

)

train\_datagen.fit(x\_train)

val\_datagen = ImageDataGenerator(preprocessing\_function = tf.keras.applications.vgg19.preprocess\_input) val\_datagen.fit(x\_val)

from keras.callbacks import ReduceLROnPlateau

learning\_rate\_reduction = ReduceLROnPlateau(monitor='val\_accuracy',

patience=3, verbose=1, factor=0.5,

min\_lr=0.00001)

# We have used only 16 layers out of 19 layers in the CNN

vgg\_model = tf.keras.applications.VGG19( include\_top=False,

weights=None,

input\_shape=(32,32,3),

)

vgg\_model.summary() Model: "vgg19"

Layer (type) Output Shape Param #

=================================================================

|  |  |  |
| --- | --- | --- |
| input\_1 (InputLayer) | [(None, 32, 32, 3)] | 0 |
| block1\_conv1 (Conv2D) | (None, 32, 32, 64) | 1792 |
| block1\_conv2 (Conv2D) | (None, 32, 32, 64) | 36928 |
| block1\_pool (MaxPooling2D) | (None, 16, 16, 64) | 0 |
| block2\_conv1 (Conv2D) | (None, 16, 16, 128) | 73856 |
| block2\_conv2 (Conv2D) | (None, 16, 16, 128) | 147584 |
| block2\_pool (MaxPooling2D) | (None, 8, 8, 128) | 0 |
| block3\_conv1 (Conv2D) | (None, 8, 8, 256) | 295168 |
| block3\_conv2 (Conv2D) | (None, 8, 8, 256) | 590080 |
| block3\_conv3 (Conv2D) | (None, 8, 8, 256) | 590080 |
| block3\_conv4 (Conv2D) | (None, 8, 8, 256) | 590080 |
| block3\_pool (MaxPooling2D) | (None, 4, 4, 256) | 0 |
| block4\_conv1 (Conv2D) | (None, 4, 4, 512) | 1180160 |
| block4\_conv2 (Conv2D) | (None, 4, 4, 512) | 2359808 |
| block4\_conv3 (Conv2D) | (None, 4, 4, 512) | 2359808 |
| block4\_conv4 (Conv2D) | (None, 4, 4, 512) | 2359808 |
| block4\_pool (MaxPooling2D) | (None, 2, 2, 512) | 0 |

|  |  |  |
| --- | --- | --- |
| block5\_conv1 (Conv2D) | (None, 2, 2, 512) | 2359808 |
| block5\_conv2 (Conv2D) | (None, 2, 2, 512) | 2359808 |
| block5\_conv3 (Conv2D) | (None, 2, 2, 512) | 2359808 |
| block5\_conv4 (Conv2D) | (None, 2, 2, 512) | 2359808 |
| block5\_pool (MaxPooling2D) | (None, 1, 1, 512) | 0 |

=================================================================

Total params: 20024384 (76.39 MB)

Trainable params: 20024384 (76.39 MB)

Non-trainable params: 0 (0.00 Byte)

model = tf.keras.Sequential() model.add(vgg\_model)

model.add(Flatten())

model.add(Dense(1024, activation = 'relu')) model.add(BatchNormalization())

model.add(Dense(1024, activation = 'relu')) model.add(BatchNormalization())

model.add(Dense(256, activation = 'relu')) model.add(BatchNormalization())

model.add(Dropout(0.5))

model.add(Dense(100, activation = 'softmax')) model.summary()

Model: "sequential"

Layer (type) Output Shape Param #

=================================================================

|  |  |  |
| --- | --- | --- |
| vgg19 (Functional) | (None, 1, 1, 512) | 20024384 |
| flatten (Flatten) | (None, 512) | 0 |
| dense (Dense) | (None, 1024) | 525312 |
| batch\_normalization (Batch Normalization) | (None, 1024) | 4096 |
| dense\_1 (Dense) | (None, 1024) | 1049600 |
| batch\_normalization\_1 (Bat chNormalization) | (None, 1024) | 4096 |
| dense\_2 (Dense) | (None, 256) | 262400 |
| batch\_normalization\_2 (Bat chNormalization) | (None, 256) | 1024 |
| dropout (Dropout) | (None, 256) | 0 |
| dense\_3 (Dense) | (None, 100) | 25700 |

=================================================================

Total params: 21896612 (83.53 MB)

Trainable params: 21892004 (83.51 MB)

Non-trainable params: 4608 (18.00 KB)

optimizer = tf.keras.optimizers.SGD(learning\_rate = 0.001, momentum = 0.9) model.compile(optimizer= optimizer,

loss='categorical\_crossentropy', metrics=['accuracy'])

history = model.fit(

train\_datagen.flow(x\_train, y\_train, batch\_size = 128),

validation\_data = val\_datagen.flow(x\_val,y\_val, batch\_size = 128), epochs = 10,

verbose = 1,

callbacks = [learning\_rate\_reduction]

)

Epoch 1/10

313/313 [==============================] - 47s 107ms/step - loss: 5.1790 - accuracy: 0.0158 - val\_loss: 4.7704 - val\_accuracy: 0.02 Epoch 2/10

313/313 [==============================] - 32s 101ms/step - loss: 4.8395 - accuracy: 0.0225 - val\_loss: 4.3828 - val\_accuracy: 0.03

Epoch 3/10

313/313 [==============================] - 32s 102ms/step - loss: 4.7381 - accuracy: 0.0245 - val\_loss: 4.3177 - val\_accuracy: 0.03 Epoch 4/10

313/313 [==============================] - 32s 101ms/step - loss: 4.6531 - accuracy: 0.0262 - val\_loss: 4.2949 - val\_accuracy: 0.03 Epoch 5/10

313/313 [==============================] - 32s 101ms/step - loss: 4.5842 - accuracy: 0.0277 - val\_loss: 4.2602 - val\_accuracy: 0.04 Epoch 6/10

313/313 [==============================] - 31s 99ms/step - loss: 4.5129 - accuracy: 0.0295 - val\_loss: 4.3661 - val\_accuracy: 0.029 Epoch 7/10

313/313 [==============================] - 33s 104ms/step - loss: 4.4780 - accuracy: 0.0286 - val\_loss: 4.2673 - val\_accuracy: 0.03 Epoch 8/10

313/313 [==============================] - ETA: 0s - loss: 4.4381 - accuracy: 0.0283

Epoch 8: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.

313/313 [==============================] - 32s 103ms/step - loss: 4.4381 - accuracy: 0.0283 - val\_loss: 4.4033 - val\_accuracy: 0.02 Epoch 9/10

313/313 [==============================] - 31s 99ms/step - loss: 4.3929 - accuracy: 0.0291 - val\_loss: 4.1842 - val\_accuracy: 0.040 Epoch 10/10

313/313 [==============================] - 32s 101ms/step - loss: 4.3699 - accuracy: 0.0294 - val\_loss: 4.1782 - val\_accuracy: 0.04

acc = history.history['accuracy']

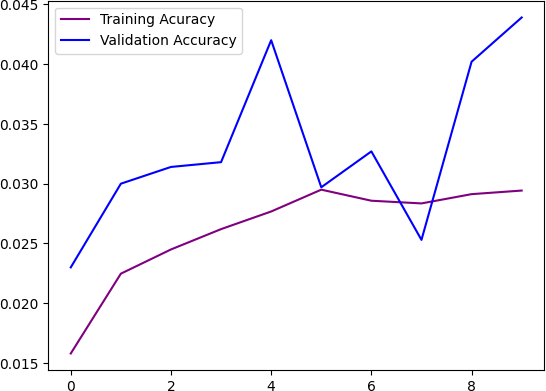
val\_acc = history.history['val\_accuracy']

plt.figure()

plt.plot(acc,color = 'purple',label = 'Training Acuracy')

plt.plot(val\_acc,color = 'blue',label = 'Validation Accuracy') plt.legend()

<matplotlib.legend.Legend at 0x7a06b02c0400>



loss = history.history['loss']

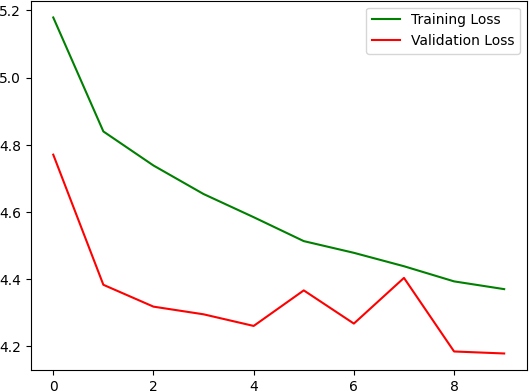
val\_loss = history.history['val\_loss']

plt.figure()

plt.plot(loss,color = 'green',label = 'Training Loss')

plt.plot(val\_loss,color = 'red',label = 'Validation Loss') plt.legend()

<matplotlib.legend.Legend at 0x7a06b01b23e0>



X\_test = tf.keras.applications.vgg19.preprocess\_input(X\_test) y\_pred = np.argmax(model.predict(X\_test), axis=-1)

y\_pred[:10]

313/313 [==============================] - 4s 9ms/step array([85, 75, 85, 43, 53, 75, 85, 75, 52, 82])

from sklearn.metrics import confusion\_matrix, accuracy\_score print('Testing Accuarcy : ', accuracy\_score(Y\_test, y\_pred))

Testing Accuarcy : 0.042

cm = confusion\_matrix(Y\_test, y\_pred) cm

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| array([[ 0, | 0, | 3, ..., | 0, | 2, | 0], |
| [ 0, | 0, | 4, ..., | 0, | 0, | 0], |
| [ 0,  ..., | 0, | 9, ..., | 0, | 7, | 0], |
| [ 0, | 0, | 0, ..., | 0, | 0, | 0], |
| [ 0, | 0, | 10, ..., | 0, | 1, | 0], |
| [ 0, | 0, | 1, ..., | 0, | 2, | 0]]) |

import itertools

def plot\_confusion\_matrix(cm, classes,

normalize=True,

title='Confusion matrix', cmap=plt.cm.Greens):

"""

This function prints and plots the confusion matrix.

Normalization can be applied by setting `normalize=True`. """

plt.imshow(cm, interpolation='nearest', cmap=cmap) plt.title(title)

plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=30) plt.yticks(tick\_marks, classes)

if normalize:

cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis] print("Normalized confusion matrix")

else:

print('Confusion matrix, without normalization') #print(cm)

thresh = cm.max() / 2.

for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])): plt.text(j, i, cm[i, j],

horizontalalignment="center",

color="white" if cm[i, j] > thresh else "black")

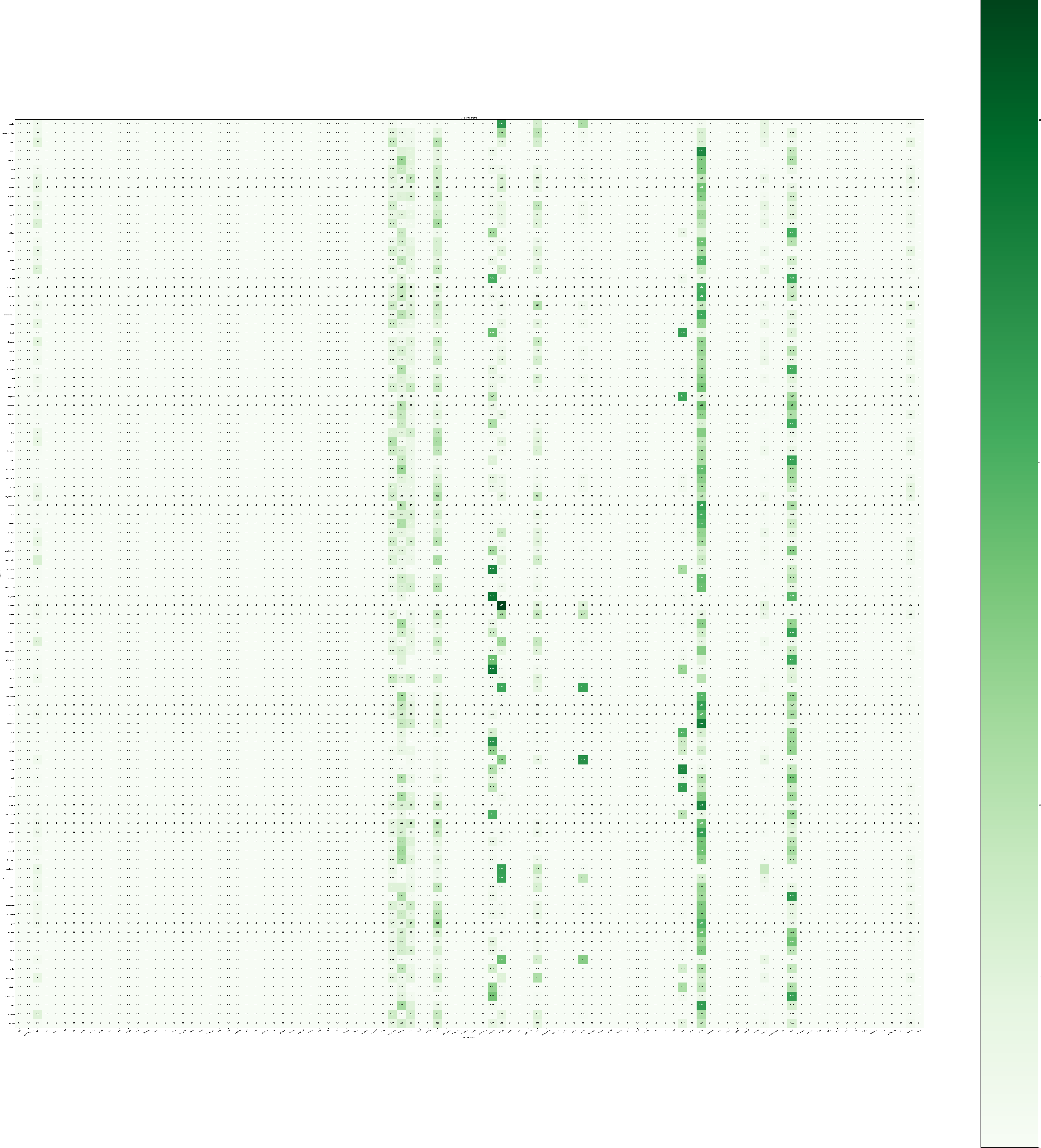
plt.tight\_layout()

plt.ylabel('True label')

plt.xlabel('Predicted label')

plt.figure(figsize=(100,100))

plot\_confusion\_matrix(cm,classes)

Normalized confusion matrix