**Practical Assignment**

**Objective: - Image Classification with CIFAR 100**

This dataset is just like the CIFAR-10, except it has 100 classes containing 600 images each. There are 500 training images and 100 testing images per class. The 100 classes in the CIFAR-100 are grouped into 20 superclasses. Each image comes with a "fine" label (the class to which it belongs) and a "coarse" label (the superclass to which it belongs).

**Dataset Link: -**[**https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz**](https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz)

The dataset is not direct images. Please decode it using your own techniques.

**Task: -** Create a Web Application using Flask. Use the end user should be able to upload an image and get results with the prediction score.

**Deployment: -** Any Free Platform(Try to look out for free options.)

**Assignment Submission: -** Only submit the hosted app link.

**import** os

print(os**.**environ['CONDA\_DEFAULT\_ENV'])

**import** keras

**import** tensorflow **as** tf

config **=** tf**.**ConfigProto(device\_count**=**{"CPU": 4})

keras**.**backend**.**tensorflow\_backend**.**set\_session(tf**.**Session(config**=**config))

base

Using TensorFlow backend.

**%matplotlib** inline

**import** warnings

warnings**.**filterwarnings("ignore")

*# basic requirements*

**import** cv2

**import** numpy **as** np

**import** pickle

**import** pandas **as** pd

*# visualization*

**import** matplotlib.pyplot **as** plt

*# For creating mode*

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

**from** sklearn.preprocessing **import** LabelEncoder

**import** keras

**from** keras.datasets **import** cifar100

**from** keras.models **import** Sequential

**from** keras.layers.core **import** Dense, Dropout, Activation, Flatten,Lambda

**from** keras.preprocessing.image **import** ImageDataGenerator

**from** keras.layers **import** Conv2D, MaxPool2D,Conv2D, MaxPooling2D, BatchNormalization

**from** keras **import** optimizers

**from** keras.utils **import** to\_categorical,plot\_model

**from** keras **import** backend **as** K

**from** keras **import** **\***

**from** keras.optimizers **import** **\***

**from** keras.callbacks **import** **\***

**from** keras.layers **import** **\***

**The CIFAR-100 dataset**

This dataset is just like the CIFAR-10, except it has 100 classes containing 600 images each. There are 500 training images and 100 testing images per class. The 100 classes in the CIFAR-100 are grouped into 20 superclasses. Each image comes with a "fine" label (the class to which it belongs) and a "coarse" label (the superclass to which it belongs).  
The images and labels are all taken from the CIFAR-100 dataset which was collected by **Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton**. The 100 object class labels are

| **Superclass** | **Classes** |
| --- | --- |

| aquatic mammals |beaver, dolphin, otter, seal, whale | fish |aquarium fish, flatfish, ray, shark, trout | flowers |orchids, poppies, roses, sunflowers, tulips | food containers |bottles, bowls, cans, cups, plates | fruit and vegetables |apples, mushrooms, oranges, pears, sweet peppers | household electrical devices |clock, computer keyboard, lamp, telephone, television | household furniture |bed, chair, couch, table, wardrobe | insects |bee, beetle, butterfly, caterpillar, cockroach | large carnivores |bear, leopard, lion, tiger, wolf | large man-made outdoor things |bridge, castle, house, road, skyscraper | large natural outdoor scenes |cloud, forest, mountain, plain, sea | large omnivores and herbivores |camel, cattle, chimpanzee, elephant, kangaroo | medium-sized mammals |fox, porcupine, possum, raccoon, skunk | non-insect invertebrates |crab, lobster, snail, spider, worm | people |baby, boy, girl, man, woman | reptiles |crocodile, dinosaur, lizard, snake, turtle | small mammals |hamster, mouse, rabbit, shrew, squirrel | trees |maple, oak, palm, pine, willow | vehicles 1 |bicycle, bus, motorcycle, pickup truck, train | vehicles 2 |lawn-mower, rocket, streetcar, tank, tractor

# Loading Data

**with** open("data/train","rb") **as** f:

df **=** pickle**.**load(f,encoding**=**'bytes')

### Getting images

features **=** df[b'data']

print(features**.**shape)

*# reshaping data from from flat to 32\*32\*3*

features **=** features**.**reshape((len(df[b'data']), 3, 32, 32))

features **=** features**.**transpose(0, 2, 3, 1)

#### Here are some images

f,ax **=** plt**.**subplots(2,5,figsize**=**(8,3))

f **=** plt**.**figure(figsize**=**(20,20))

**for** i **in** range(5):

**for** j **in** range(2):

ax[j][i]**.**imshow(features[np**.**random**.**randint(6000)])

plt**.**show()

### Getting labes

*# getting fine-labes of images (in the form of numberical vales from 0-99)*

fine\_labels **=** np**.**array(df[b'fine\_labels'])

*# getting coarse-labes of images (in the form of numerical values from 0-19)*

coarse\_labels **=** np**.**array(df[b'coarse\_labels'])

*# getting file\_name (in the form of Name-tag)*

file\_name **=** df[b'filenames']

### Extracting Name lables from filename

name\_label **=** []

**for** file **in** file\_name:

stripped **=** str(file)**.**split("\_s\_")[0]

name\_label**.**append(stripped)

print("SOME LABEL NAMES: ", name\_label[:40])

## About Data

50000 labelled examples (500 per class) are provided for training, with a further 10000 examples (100 per class) used for testing.  
Each images has 3 RGB colour channels and pixel dimensions 32×32 for an overall size per input of 3×32×32=3072.

print("No of images in CIFAR-100 dataset:", features**.**shape[0] , " \n")

print("Dimensions of image:", features[0]**.**shape , " \n")

print("Total number of fine-categories:", len(set(fine\_labels)) , " \n")

print("Total number of coarse-categories:", len(set(coarse\_labels)) , " \n")

print("Total number of name-labes:", len(set(name\_label)) , " \n")

### Visualisation of original data with labels

f,ax **=** plt**.**subplots(2,5,figsize**=**(9,4))

**for** i **in** range(10):

plt**.**subplot(2,5,i**+**1)

plt**.**tight\_layout()

plt**.**imshow(features[i],cmap **=**'gray')

plt**.**title(name\_label[i])

plt**.**show()

# Data pre processing

### Train Test Split

#### Why Training and Testing Data Sets?

Separating data into training and testing sets is an important part of evaluating data mining models. Typically, when you separate a data set into a training set and testing set, most of the data is used for training, and a smaller portion of the data is used for testing. Analysis Services randomly samples the data to help ensure that the testing and training sets are similar. By using similar data for training and testing, you can minimize the effects of data discrepancies and better understand the characteristics of the model.  
  
After a model has been processed by using the training set, you test the model by making predictions against the test set. Because the data in the testing set already contains known values for the attribute that you want to predict, it is easy to determine whether the model's guesses are correct. [more](https://docs.microsoft.com/en-us/sql/analysis-services/data-mining/training-and-testing-data-sets?view=sql-server-2017)

features **=** features

fine\_labels **=** fine\_labels

x\_train, x\_test, y\_train, y\_test **=** train\_test\_split(features, fine\_labels,

test\_size **=** 0.1,

stratify **=** fine\_labels)

print("Train shape (input):", x\_train**.**shape)

print("Test shape (input):", x\_test**.**shape)

print("Train shape (labels):", y\_train**.**shape)

print("Test shape (labels):", y\_test**.**shape)

### Normalization of Data

WHY: If we didn't scale our input training vectors, the ranges of our distributions of feature values would likely be different for each feature, and thus the learning rate would cause corrections in each dimension that would differ (proportionally speaking) from one another. We might be over compensating a correction in one weight dimension while undercompensating in another. [SOURCE](https://stats.stackexchange.com/questions/185853/why-do-we-need-to-normalize-the-images-before-we-put-them-into-cnn)

x\_train **=** x\_train**.**astype(np**.**float32)

x\_test **=** x\_test**.**astype(np**.**float32)

(x\_train, y\_train), (x\_test, y\_test) **=** cifar100**.**load\_data()

**def** normalize(X\_train,X\_test):

*# tonormalize inputs for zero mean and unit variance*

mean **=** np**.**mean(X\_train,axis**=**(0,1,2,3))

std **=** np**.**std(X\_train, axis**=**(0, 1, 2, 3))

X\_train **=** (X\_train**-**mean)**/**(std**+**1e-7)

X\_test **=** (X\_test**-**mean)**/**(std**+**1e-7)

print("Standrad deviation",std," \nMean",mean)

**return** X\_train, X\_test

x\_train, x\_test **=** normalize(x\_train,x\_test)

### Encoding of labels

This means that categorical data must be converted to a numerical form. If the categorical variable is an output variable, you may also want to convert predictions by the model back into a categorical form in order to present them or use them in some application.

y\_train **=** to\_categorical(y\_train)**.**astype(np**.**uint8)

y\_test **=** to\_categorical(y\_test)**.**astype(np**.**uint8)

print("Sahpe after encoding:",y\_train**.**shape, y\_test**.**shape)

## Creating model

### Some constants

weight\_decay **=** 0.0005

img\_dim **=** 32

number\_of\_classes **=** 100

image\_shape **=** [img\_dim, img\_dim, 3]

**def** add\_layer(model,num,dropout**=True**):

model**.**add(Conv2D(num, (3, 3), padding**=**'same',kernel\_regularizer**=**regularizers**.**l2(weight\_decay)))

model**.**add(Activation('relu'))

model**.**add(BatchNormalization())

**if**(dropout):

model**.**add(Dropout(0.4))

**return** model

**def** baseline\_model():

model **=** Sequential()

model**.**add(Conv2D(64, (3, 3), padding**=**'same',

input\_shape**=**image\_shape,kernel\_regularizer**=**regularizers**.**l2(weight\_decay)))

model**.**add(Activation('relu'))

model**.**add(BatchNormalization())

model**.**add(Dropout(0.3))

model **=** add\_layer(model,64,dropout**=False**)

model**.**add(MaxPooling2D(pool\_size**=**(2, 2)))

model **=** add\_layer(model,128,dropout**=True**)

model **=** add\_layer(model,128,dropout**=False**)

model**.**add(MaxPooling2D(pool\_size**=**(2, 2)))

model **=** add\_layer(model,256,dropout**=True**)

model **=** add\_layer(model,256,dropout**=True**)

model **=** add\_layer(model,256,dropout**=False**)

model**.**add(MaxPooling2D(pool\_size**=**(2, 2)))

model **=** add\_layer(model,512,dropout**=True**)

model **=** add\_layer(model,512,dropout**=True**)

model **=** add\_layer(model,512,dropout**=False**)

model**.**add(MaxPooling2D(pool\_size**=**(2, 2)))

model **=** add\_layer(model,512,dropout**=True**)

model **=** add\_layer(model,512,dropout**=True**)

model **=** add\_layer(model,512,dropout**=False**)

model**.**add(MaxPooling2D(pool\_size**=**(2, 2)))

model**.**add(Dropout(0.5))

model**.**add(Flatten())

model**.**add(Dense(512,kernel\_regularizer**=**regularizers**.**l2(weight\_decay)))

model**.**add(Activation('relu'))

model**.**add(BatchNormalization())

model**.**add(Dropout(0.5))

model**.**add(Dense(number\_of\_classes))

model**.**add(Activation('softmax'))

**return** model

### Plot of the layers in the VGG-NET

model **=** baseline\_model()

plot\_model(model, to\_file**=**'vgg-net.png')

### Model

model**.**summary()

# Data Augmentation

Generate batches of tensor image data with real-time data augmentation. The data will be looped over (in batches). Rather than performing the operations on your entire image dataset in memory, the API is designed to be iterated by the deep learning model fitting process, creating augmented image data for you just-in-time. This reduces your memory overhead, but adds some additional time cost during model training.[more](https://machinelearningmastery.com/image-augmentation-deep-learning-keras/)

data\_generator **=** ImageDataGenerator(

*# randomly rotate images in the range (degrees, 0 to 180)*

rotation\_range**=**15,

*# randomly shift images horizontally (fraction of total width)*

width\_shift\_range**=**0.1,

*# randomly shift images vertically (fraction of total height)*

height\_shift\_range**=**0.1,

*# randomly flip images*

horizontal\_flip**=True**,

*# randomly flip images*

vertical\_flip**=False**,

*# set input mean to 0 over the dataset*

featurewise\_center**=False**,

*# set each sample mean to 0*

samplewise\_center**=False**,

*# divide inputs by std of the dataset*

featurewise\_std\_normalization**=False**,

*# divide each input by its std*

samplewise\_std\_normalization**=False**,

*# apply ZCA whitening*

zca\_whitening**=False**)

data\_generator**.**fit(x\_train)

### Normal Images

img\_rows, img\_cols **=** 32, 32

num\_classes **=** 100

*#The data, shuffled and split between train and test sets*

(xa\_train, ya\_train), (xa\_test, ya\_test) **=** cifar100**.**load\_data()

xa\_train **=** xa\_train**.**reshape(xa\_train**.**shape[0], img\_rows, img\_cols, 3)

xa\_test **=** xa\_test**.**reshape(xa\_test**.**shape[0], img\_rows, img\_cols, 3)

input\_shape **=** (img\_rows, img\_cols, 3)

xa\_train **=** xa\_train**.**astype('float32')

xa\_test **=** xa\_test**.**astype('float32')

xa\_train **/=** 255

xa\_test **/=** 255

*#Look at the first 9 images from the dataset*

plt**.**figure(figsize**=**(10,10))

images **=** range(0,5)

**for** i **in** images:

plt**.**subplot(550 **+** 1 **+** i)

plt**.**imshow(xa\_train[i], cmap**=**plt**.**get\_cmap('gray'))

*#Show the plot*

plt**.**show()

### Image flip (horizontal\_flip)

*# Flip images vertically*

datagen **=** ImageDataGenerator(horizontal\_flip**=True**)

*# fit parameters from data*

datagen**.**fit(xa\_train)

*# Configure batch size and retrieve one batch of images*

plt**.**figure(figsize**=**(10,10))

**for** Xa\_batch, yabatch **in** datagen**.**flow(xa\_train, ya\_train, batch\_size**=**9):

*# Show 9 images*

**for** i **in** range(0, 5):

plt**.**subplot(550 **+** 1 **+** i)

plt**.**imshow(Xa\_batch[i]**.**reshape(img\_rows, img\_cols, 3))

*# show the plot*

plt**.**show()

**break**

### Image rotation

randomly rotate images in the range (degrees, 0 to 180)

datagen **=** ImageDataGenerator(rotation\_range**=**180)

*# fit parameters from data*

datagen**.**fit(xa\_train)

*# Configure batch size and retrieve one batch of images*

plt**.**figure(figsize**=**(10,10))

**for** X\_batch, y\_batch **in** datagen**.**flow(xa\_train, ya\_train, batch\_size**=**9):

*# Show 9 images*

**for** i **in** range(0, 5):

plt**.**subplot(550 **+** 1 **+** i)

plt**.**imshow(X\_batch[i]**.**reshape(img\_rows, img\_cols, 3))

*# show the plot*

plt**.**show()

**break**

### Width shift range

randomly shift images horizontally (fraction of total width)

datagen **=** ImageDataGenerator(width\_shift\_range**=**0.1)

*# fit parameters from data*

datagen**.**fit(xa\_train)

*# Configure batch size and retrieve one batch of images*

plt**.**figure(figsize**=**(10,10))

**for** X\_batch, y\_batch **in** datagen**.**flow(xa\_train, ya\_train, batch\_size**=**9):

*# Show 9 images*

**for** i **in** range(0, 5):

plt**.**subplot(550 **+** 1 **+** i)

plt**.**imshow(X\_batch[i]**.**reshape(img\_rows, img\_cols, 3))

*# show the plot*

plt**.**show()

**break**

### Height shift range

randomly shift images vertically (fraction of total height)

datagen **=** ImageDataGenerator(height\_shift\_range**=**0.1)

*# fit parameters from data*

datagen**.**fit(xa\_train)

*# Configure batch size and retrieve one batch of images*

plt**.**figure(figsize**=**(10,10))

**for** X\_batch, y\_batch **in** datagen**.**flow(xa\_train, ya\_train, batch\_size**=**9):

*# Show 9 images*

**for** i **in** range(0, 5):

plt**.**subplot(550 **+** 1 **+** i)

plt**.**imshow(X\_batch[i]**.**reshape(img\_rows, img\_cols, 3))

*# show the plot*

plt**.**show()

**break**

**Compiling The model**

BATCH\_SIZE **=** 128

EPOCHS **=** 250

learning\_rate **=** 0.1

lr\_decay **=** 1e-6

lr\_drop **=** 20

**def** lr\_scheduler(epoch):

**return** learning\_rate **\*** (0.5 **\*\*** (epoch **//** lr\_drop))

reduce\_lr **=** keras**.**callbacks**.**LearningRateScheduler(lr\_scheduler)

optimizer **=** optimizers**.**SGD(lr**=**learning\_rate, decay**=**lr\_decay, momentum**=**0.9, nesterov**=True**)

model**.**compile(loss**=**'categorical\_crossentropy', optimizer**=**optimizer,metrics**=**['accuracy'])

# Train and Testing

##### **IS\_TRAIN = True** for training mode

IS\_TRAIN **=** **False**

**if**(IS\_TRAIN**==True**):

history **=** model**.**fit\_generator(data\_generator**.**flow(x\_train, y\_train,

batch\_size**=**batch\_size),

steps\_per\_epoch**=**x\_train**.**shape[0] **//** batch\_size,

epochs**=**EPOCHS,callbacks**=**[reduce\_lr])

*# saving log into file log.txt file*

**with** opne("log.txt") **as** f:

f**.**write(history)

*# saving trained model*

model**.**save\_weights('cifar100.h5')

**else**:

*# loading trained model*

model**.**load\_weights('cifar100.h5')

## Make predictions

pred\_x **=** model**.**predict(x\_test)

*# How many predisctions are wrong*

is\_not\_equal **=** (np**.**argmax(pred\_x,1)**!=**np**.**argmax(y\_test,1))

print(is\_not\_equal)

loss **=** sum(is\_not\_equal)**/**len(is\_not\_equal)

print("Loss in prediction: ",loss)

## Training score and accuracy

train\_score, train\_acc **=** model**.**model**.**evaluate(x\_train, y\_train)

print("Training Score = ", train\_score, " \nTraining Accuracy = ", train\_acc, "%")

## Test score and accuracy

test\_score, test\_acc **=** model**.**model**.**evaluate(x\_test, y\_test)

print("Validation Score = ", test\_score, " \nValidation Accuracy =", test\_acc, "%")

## Confusion matrix

**from** sklearn.metrics **import** confusion\_matrix

**import** seaborn **as** sns

results **=** confusion\_matrix(np**.**argmax(y\_test,1), np**.**argmax(pred\_x,1))

print(results)

df\_cm **=** pd**.**DataFrame(results, index **=** [i **for** i **in** range(100)],

columns **=** [i **for** i **in** range(100)])

plt**.**figure(figsize **=** (15,15))

sns**.**heatmap(df\_cm, annot**=True**)