Image Classification documentation

- For the image classification modeling, we are using the Keras library

Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library

For the image classification we are using ResNet model to classify the images in CIFAR-10

ResNet, short for Residual Networks, is a classic neural network used **as a backbone for many computer vision tasks**. This model was the winner of the ImageNet challenge in 2015. The fundamental breakthrough with ResNet was it allowed us to train extremely deep neural networks with 150+layers successfully.

Installing the required modules

```
!pip install -q keras
!pip install image-classifiers
!pip install git+https://github.com/qubvel/classification_models.git
```

Importing The libraries

```
import keras
from keras.datasets import cifar10
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.applications.resnet50 import preprocess_input
import numpy as np
%matplotlib inline
```

```
import matplotlib.pyplot as plt
from keras import optimizers
from keras.callbacks import ModelCheckpoint
from keras.applications.vgg16 import preprocess_input, decode_predictions
from keras.preprocessing import image
import keras.backend as K
import cv2
import sys
```

After we are done importing the required library, we can write a code to load the dataset
For the image classification, we are using CIFAR-10 dataset

To use keras in python for machine learning we import it using the given code

```
import keras
```

We are using Numpy Library to convert the List into an Array in this project

```
import numpy as np
```

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays

Matplotlib library is being used to plot the graph of the accuracy and loss of the model

```
import matplotlib.pyplot as plt
```

Matplotlib is a comprehensive library for **creating static, animated, and interactive visualizations in Python**.

```
batch_size = 128
```

The batch size is **a number of samples processed before the model is updated**

```
n_classes = 10 #Because CIFAR10 has 10 classes
epochs = 20
```

The number of epochs is the number of complete passes through the training dataset. The size of a batch must be more than or equal to one and less than or equal to the number of samples in the training dataset.

After the dataset is loaded we split the dataset into Train and Test sets. This is done using the code given below

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

```
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

Training Dataset

x_train is the feature of the train dataset

y_train is the label of the train dataset

Test Dataset

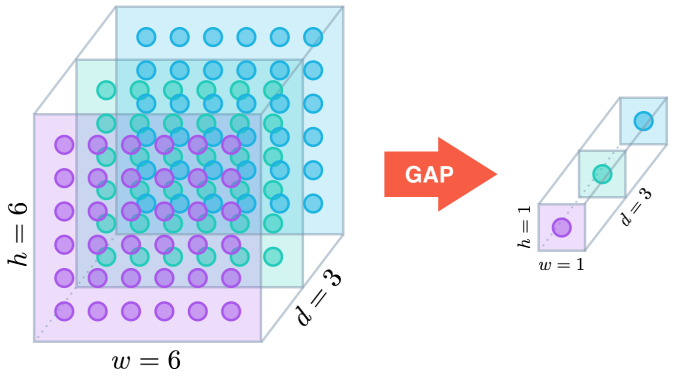
x_test is the feature of the test dataset

y_test is the label of the test dataset

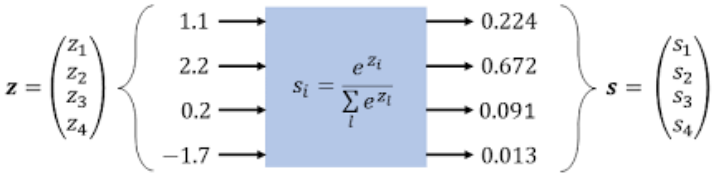
```
# Convert class vectors to binary class matrices.
y_train = keras.utils.to_categorical(y_train, n_classes)
y_test = keras.utils.to_categorical(y_test, n_classes)

print(np.mean(x_test[0]))
print(np.mean(x_train[0]))


# normalize inputs from 0-255 to 0.0-1.0
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train = x_train / 255.0
x_test = x_test / 255.0



print(np.mean(x_test[0]))
print(np.mean(x_train[0]))
```

Using the ResNet50 Architecture

```
base_model = ResNet50(input_shape=(32,32,3), weights='imagenet',
include_top=False)
```




```
x = keras.layers.GlobalAveragePooling2D()(base_model.output)
```




```
output = keras.layers.Dense(n_classes, activation='softmax')(x)
model = keras.models.Model(inputs=[base_model.input], outputs=[output])
```

Stochastic Gradient Descent (SGD)

Stochastic gradient descent is an optimization algorithm often used in machine learning applications to find the model parameters that correspond to the best fit between predicted and actual outputs

```
sgd = optimizers.SGD(lr=0.1, decay=0.0001, momentum=0.9, nesterov=True)
```

categorical_crossentropy: Used as a loss function for multi-class classification model where there are two or more output labels. The output label is assigned one-hot category encoding value in form of 0s and 1.

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

mode.summary() display the architecture of the model

```
model.summary()
```

```
scores = model.evaluate(x_test, y_test, verbose=1)
print('Test loss:', scores[0])
print('Test accuracy:', scores[1])
```

```
import math
```

```
# learning rate schedule
def step_decay(epoch):
  initial_lrate = 0.1
  drop = 0.7
  epochs_drop = 7.0
  lrate = initial_lrate * math.pow(drop,
math.floor((1+epoch)/epochs_drop))
  return lrate



for i in range(1,26):
  print("Epoch "+str(i)+ " : ",step_decay(i))
```

This function keeps the initial learning rate for the first 26 epochs and decreases it exponentially after that.

```
from keras.callbacks import LearningRateScheduler
lrate = LearningRateScheduler(step_decay)

# checkpoint
filepath="Assignment_5.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_acc', verbose=1,
save_best_only=True, mode='max')
```

Model Training

```
model.fit(x_train, y_train, batch_size=batch_size,
validation_data=(x_test, y_test), epochs=epochs,
verbose=1,callbacks=[lrate,checkpoint])

scores = model.evaluate(x_test, y_test, verbose=1)
print('Test loss:', scores[0])
print('Test accuracy:', scores[1])
```

Saving the Model Weight

```
# Save the trained weights in to .h5 format
model.save_weights("ResNet_CIFAR10_Weights_Best.h5")
print("Saved model to disk")
```

Model Saving

```
# Save the trained model in to .h5 format
model.save("ResNet_CIFAR10_Model_Best.h5")
print("Saved model to disk")
```

Predicting the 10000 images

```
y_pred = model.predict(x_test, verbose = 1)
```

The maximum value along a given axis.

```
y_pred_classes = np.argmax(y_pred, axis=1)
y_pred_max_probas = np.max(y_pred, axis=1)
```

Loading the saved model

```
model = keras.models.load_model('/content/ResNet_CIFAR10_Model_Best.h5')
```

Predicting the first image

```
model.predict(np.array([x_test[0]]))
```