CHAPTER FOUR

SYSTEM IMPLEMENTATION AND RESULTS

4.1 PREAMBLE

4.2 INSTALLATION REQUIREMENTS

The hardware (physical components of a computer system that can be seen, touched or felt) and software (both system software and the application software installed and used in the system development) tools needed to satisfy these objectives highlighted below:

4.2.1 Hardware Requirements

The hardware requirement includes:

A laptop or desktop computer (Preferably 64bit)

Random Access Memory (RAM): 8 Gigabytes Minimum

Processor: Intel Core i5, 2.4 GHz Minimum

4.2.2 Software Requirements

The software requirements for the development of this system include:

Windows Operating System (8/10)

JetBrains PyCharm Professional 2019

4.3 MODEL DEVELOPMENT

4.3.1 Data Collection

**The CIFAR-100 dataset**

 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).  
Here is the list of classes in the CIFAR-100:

|  |  |
| --- | --- |
| **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 |

NB: For ease of categorization, mushroom is under vegetables and bears are under carnivores even when they really aren’t.

4.3.2 Data Analysis

### Dataset layout

The archive contains the files data\_batch\_1, data\_batch\_2, ..., data\_batch\_5, as well as test\_batch. Each of these files is a Python "pickled" object produced with cPickle.

When loaded, each of the batch files contains a dictionary with the following elements:

* **data** -- a 10000x3072 [numpy](http://numpy.scipy.org/) array of uint8s. Each row of the array stores a 32x32 colour image. The first 1024 entries contain the red channel values, the next 1024 the green, and the final 1024 the blue. The image is stored in row-major order, so that the first 32 entries of the array are the red channel values of the first row of the image.
* **labels** -- a list of 10000 numbers in the range 0-9. The number at index *i* indicates the label of the *i*th image in the array **data**.

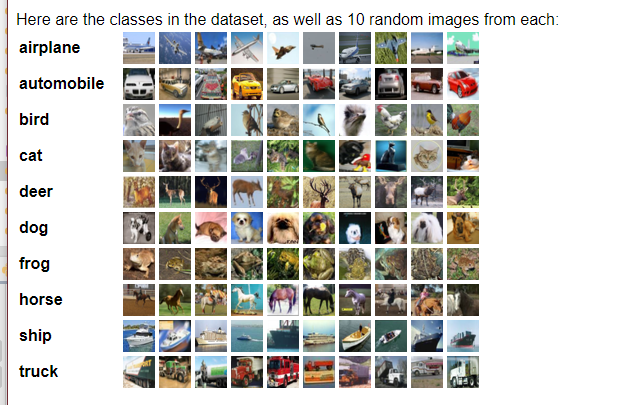
The dataset contains another file, called batches.meta. It too contains a Python dictionary object. It has the following entries:

* **label\_names** -- a 10-element list which gives meaningful names to the numeric labels in the **labels** array described above. For example, label\_names[0] == "airplane", label\_names[1] == "automobile", etc.

Link - <https://www.cs.toronto.edu/~kriz/cifar.html>

* **Source code**: [tfds.image\_classification.cifar.Cifar100](https://github.com/tensorflow/datasets/tree/master/tensorflow_datasets/image_classification/cifar.py)
* **Versions**:
  + **3.0.2** (default): No release notes.
* **Download size**: 160.71 MiB
* **Citation**:

@TECHREPORT{Krizhevsky09learningmultiple,  
    author = {Alex Krizhevsky},  
    title = {Learning multiple layers of features from tiny images},  
    institution = {},  
    year = {2009}  
}



4.3.3 Data Preprocessing

Step1

Cast a data for train to a specified dtype float32.

Cast a data for test to a specified dtype float32.



Step2

*## Normalizing the data*

Divide the test and train datasets by 255



Step3

One-Hot Encode

np.utils.to\_categorical is used to convert array of labeled data(from 0 to nb\_classes-1) to one-hot vector.



4.3.3 Model Development

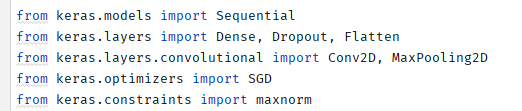
**CNN with TensorFlow — Defining Layers**

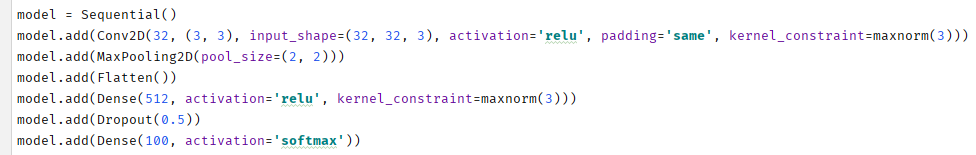
The architecture will be like this:

* 1st Convolutional Layer with 32 filters, ‘relu’ activation, ‘same’ padding, and 3 kernel constarints
* Max pooling with a pool size of 2,2
* Flatten Layer
* Dense with 512 units, ‘relu’ activation, and 3 kernel constraints
* DropOut with a 0.5 rate
* Dense with 100 units and a softmax activation

**Google each of them and write a short piece of about three sentences each to explain**

From a bigger picture, a CNN architecture accomplishes 2 major tasks: feature extraction (convolution + pooling layers) and classification.



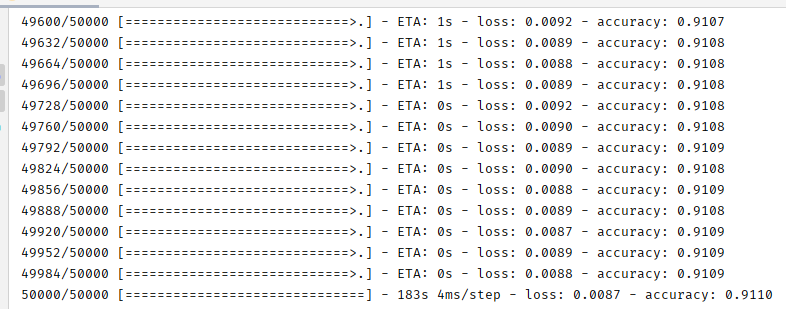




4.3.4 Model Accuracy

We split our train data in two: a training set and a validation set. Therefore, we can check the accuracy of the model train made from the ‘training set’, on the validation set.

CNN Model Accuracy



Therefore 91% accuracy and 0.8% loss

4.4 GENERAL WORKING OF THE SYSTEM

The System has been developed on a desktop application. The graphical user interface of the application was developed using Kivy application designer package and was programmed in python.

The Software has the following pages:

4.4.1 The Home Page

