KANTIPUR ENGINEERING COLLEGE

(Affiliated to Tribhuvan University)

Dhapakhel, Lalitpur

[Subject Code: CT755]

A MAJOR PROJECT MID-TERM REPORT ON

IMAGE CLASSIFICATION USING CONVOLUTION NEURAL NETWORK

Submitted by:

Aabhushan Pokharel	[45/BCT/2071]
Prabin Bhusal	[63/BCT/2071]
Rajesh Mahato	[67/BCT/2071]

A MAJOR PROJECT SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE DEGREE OF BACHELOR IN COMPUTER ENGINEERING

Submitted to:

Department of Computer and Electronics Engineering

July, 2018

IMAGE CLASSIFICATION USING CONVOLUTION NEURAL NETWORK

Submitted by:

Aabhushan Pokharel	**[45/BCT/2071]**
Prabin Bhusal	**[63/BCT/2071]**
Rajesh Mahato	**[67/BCT/2071]**

A MAJOR PROJECT SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE DEGREE OF BACHELOR IN COMPUTER ENGINEERING

Submitted to:
Department of Computer and Electronics Engineering
Kantipur Engineering College
Dhapakhel, Lalitpur

July, 2018

ABSTRACT

The number of images present in the world wide web is expanding rapidly with millions of uploads daily. This gives rise to the problem of classifying these images in appropriate categories. Convolutional Neural Networks are special neural networks that are suited to this task. This report corresponds to the impelementation of a convolutional neural network to classify images. This implementation uses four convolutional layers with Rectified Linearity Unit (ReLU) to extract feature map. The outcome of the network is a list of probability for each class. This document provides a brief introduction to convolutional neural networks, and details the progress made in this major project since the last mid-term defense.

Keywords– Image Classification, Convolutional Neural Network

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

1.1 Background

The human visual cortex is capable of recognizing and classifying a vast amount of objects that it encounters. The visual cortex is a collection of neurons connected in a certain way. A particular neuron is activated once a synapse fires depending on whether the threshold of activation was met. Neural Networks are based on this principle with weights being the analogue to the synapses. So it was natural that neural networks would be the preferred choice for image classification through object recognition.

While neural networks provided a methodology for image classification, the accuracy left room for improvement. So a new type of neural network was developed which was better suited to take images as inputs. This neural network was the Convolutional Neural Network or CovNet. This neural network has four stages including convolution, non-linearity or Rectified Non-Linearity (ReLU), pooling and fully connected layer which is a traditional multilayer perceptron. The convolution stage extracts the features from the image, ReLU introduces non-linearity as all real world images are non-linear, and pooling step reduces the dimensionality of the feature map. The fully connected layer takes the output from the above steps and classifies the image according to training dataset.

The output of the CovNet is a list of probabilities assigned to each class. The CovNet classifies the image according to the highest assigned probability. So, if an image contains a dog, a CovNet would assign a high probability of 0.94 to the dog while only 0.06 to a cat in a two class training dataset. The probability is assigned according to the weights which are adjusted by the CovNet once the inputs are initialized.

1.2 Problem Statement

The number of images online is ever growing. The number of pictures uploaded to the web is increasing day by day. The improvement in cellphone camera technology has only added to this proliferation. It has reached such an extent that it becomes impossible for anyone to manually classify these images. If an automated system that can detect the object in the image and classify them is built, then this problem could be solved.

1.3 Objective

- To implement convolutional neural network to build an image classification system

1.4 Application

- Automated image organization
- In stock photography websites for effective tagging
- In large image databases

1.5 Features

- Possible to classify different images in upto 10 classes

1.6 Feasibility Analysis

To investigate and analyze the existing procedures and to establish the key system requirements of the system, feasibility studies were conducted before starting the project. Several alternatives were explored to determine the feasibility of the project. The breakdown of the analysis is as follows.

1.6.1 Economic Feasibility

Economic feasibility determines if the project is economically viable. Since there are no hardware components to be purchased for this project, the project is viable. The project can be completed within reasonable resources as money need not be spent to complete it.

1.6.2 Technical Feasibility

The technical implementation of the software being developed is covered by existing technology. Also the system can be upgraded later as it is being built under software engineering principles. Furthermore, the required resources are available as open source as well as the existence of an online community also helps in this regard. So the project is technically feasible.

1.6.3 Operational Feasibility

The algorithm for the system being built has been implemented before by other developers and tested rigorously. The team working on the project will also be putting the system under test with different inputs to determine the operational feasibility. The system can be trained using CPU and GPU. The process is comparatively faster in GPU; however, for development, the processors in laptop computers are sufficient.

1.7 System Requirements

1.7.1 Hardware Requirements

- Intel i5 processor (recommended)
- 8GB RAM (recommended)
- Nvidia GPU (optional)

1.7.2 Software Requirements

- Windows 7/8/10 64 bit or Mac OS X Lion or higher
- Python 3.5
- Keras 2.1 library
- OpenCV library
- SKlearn library
- Numpy library

CHAPTER 2
LITERATURE REVIEW

2.1 AlexNet

In 2012, Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton released AlexNet which became the basis for most works based on Convolutional Neural Network henceforth. It won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) by a large margin in 2012.

The AlexNet used a subset ImageNet dataset which included 15 million labeled high resolution images belonging to roughly 22,000 categories. However, ILSVRC in which the algorithm was implemented used 1.2 million training images, 50,000 validation images and 150,000 testing images with 1000 images in 1000 categories.

The AlexNet architecture contains eight layers with weights. The first five layers are convolutional layers and the remaining three are fully connected. The output of the last layers is fed to a 1000 way softmax which ultimately classifies the images under the 1000 different labels. Since the training dataset included 1.2 million images, a single GPU was not sufficient. Therefore, AlexNet spread the net across two GPUs. The GPU they used was GTX 580 which had 3GB of memory. They exploited the parallelism offered by the GPU by putting half the neurons in each GPU and letting them communicate only in certain layers. They found the training time to be slightly increased with two GPUs instead of one.

They also used local normalization in addition to ReLU to add to the generalization in certain layers. Denoting by $\mathrm{a}^{\mathrm{i}}{}_{\mathrm{x,y}}$ the activity of a neuron computed by applying kernel *i* at position *(x,y)* and then applying the ReLU nonlinearity, the response normalized activity $\mathrm{b}^{\mathrm{i}}{}_{\mathrm{x,y}}$ is given by the expression

$b^{i}_{(x,y)} = a^{i}_{(x,y)}/(k+\alpha \sum\limits_{j=max(0,i-n/2)}^{min(N-1,i+n/2)} (a^{j}_{(x,y)})^2)^\beta$ where the sum runs over n adjacent kernel maps at the same spatial position and N is the total number of kernels in the layer. The ordering of kernel layer is arbitrary and determined before training begins. This sort of response normalization implements a form of lateral inhibition inspired by the type found in real neurons, creating competition for big activities among neuron outputs computed using different kernels. The constants k, n, , and are hyper-parameters whose values are determined using a validation set; they used $k = 2$, $n = 5$, $\alpha = 10^{-4}$ and $\beta = 0.75$.[1]

The contribution of AlexNet to convolutional neural network study were as follows:

- Use of rectified linear units (ReLU) as non-linearities
- Use of dropout technique to selectively ignore single neuron during training, a way to avoid overfitting the model
- Overlapping max pooling, avoiding the averaging effects of average pooling
- Use of GPUs NVIDIA GTX 580 to reduce training time [2]

2.2 VGG

The VGG networks from Oxford were the first to use much smaller 3X3 filters in each convolutional layers and also combined them as a sequence of convolutions. This was a departure from the principles of LeNet and the 9X9 or 11X11 filters of AlexNet. The advantage of VGG was that multiple 3X3 convolution in sequence could approximate the effects of larger filters such as those used by AlexNet. For example: using a 3X3 filter with stride 1 pixel, two 3X3 filters could emulate the receptive field of a 5X5 filter.

The input to the to this CovNets was a fixed-size 224 X 224 RGB image. The image was passed through a stack of convolution layers using filters with very small receptive field: 3X3. The convolution stride was fixed to 1 pixel. Five max pooling layers were used; however, not all convolution layers were followed by a pooling layer. Pooling was done via a 2X2 pixel window with stride 2. A stack of convolution layers were followed by three Fully Connected layers. The final layers was the softmax layer. The VGG networks (except one) did not contain Local Response Normalization as it did not improve performance on the ILSVRC dataset but led to increased memory consumption and computation time. The main takeway from this was the use of smaller receptive fields to approximate larger ones allowing for more filters to be used for feature extraction. This insight has been used in the models that followed the VGG. [3]

CHAPTER 3
METHODOLOGY

3.1 Software Development Model

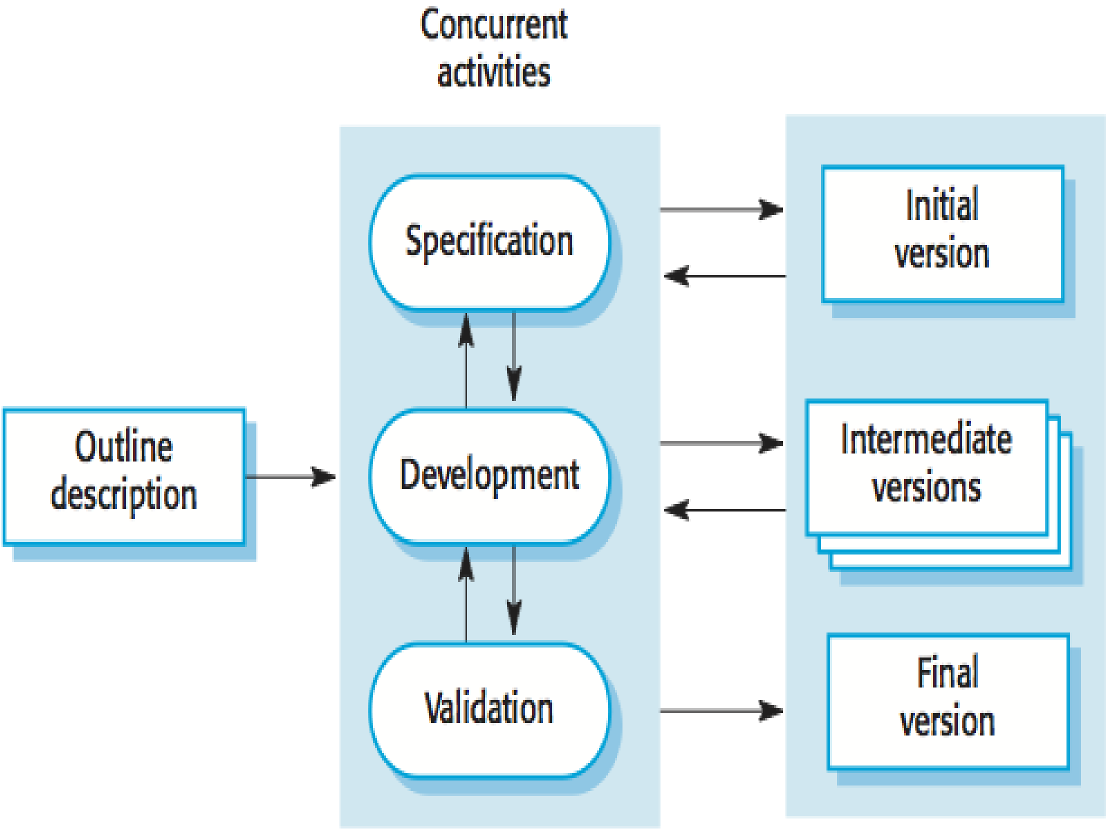


Figure 3.1: Incremental Model of Software Development[4]

The development model to be used is the incremental model of software development. Incremental model is a process of software development where requirements are broken down into multiple standalone modules of software development cycle. Each iteration passes through requirements, design, coding and testing phases. And each subsequent release of the system adds function to the previous release until all the designed functionality has been implemented. So the characteristics of this model include:

1. Development is broken down into many mini developmental works
2. Partial systems are built and functionality subsequently added
3. High priority requirements are tackled first

3.2 Algorithm

The following algorithm shows the basic steps performed by the system:

Step 1: Start

Step 2: Load data

Step 3: Define the image pixel, channel, number of classes and epochs

Step 4: Perform data pre-processing

a. resize image to standard size
b. normalize the data

Step 5: Define architecture of the network

Step 6: Train the model

Step 7: Visualize and save model configuration

Step 8: Generate plot of loss and accuracy

Step 9: Predict the class

Step 10: Stop

3.3 Flowchart

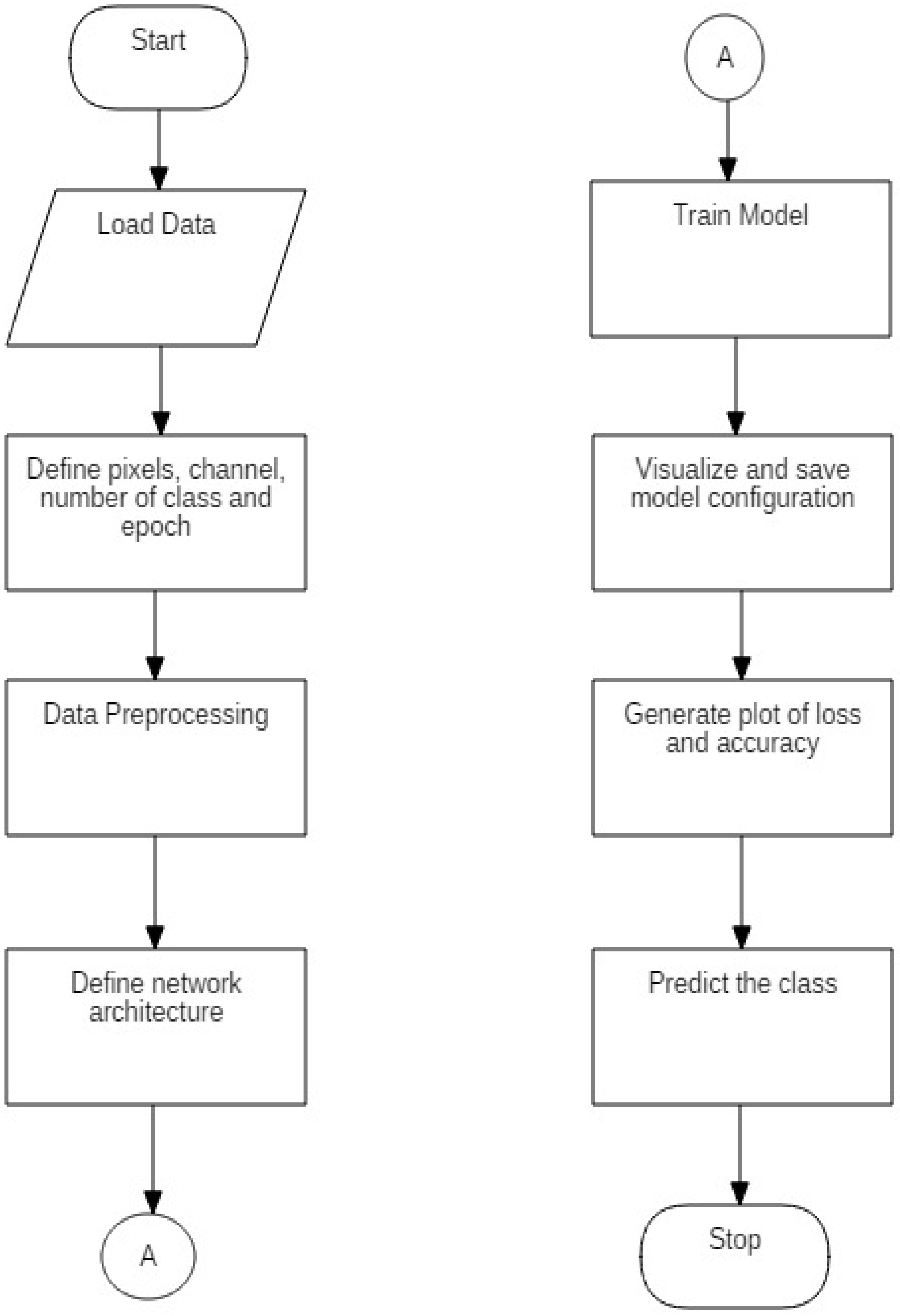

Start
Load Data
Define pixels, channel, number of class and epoch
Data Preprocessing
Define network architecture
A
A
Train Model
Visualize and save model configuration
Generate plot of loss and accuracy
Predict the class
Stop

Figure 3.2: Flowchart for the process

3.4 Architecture

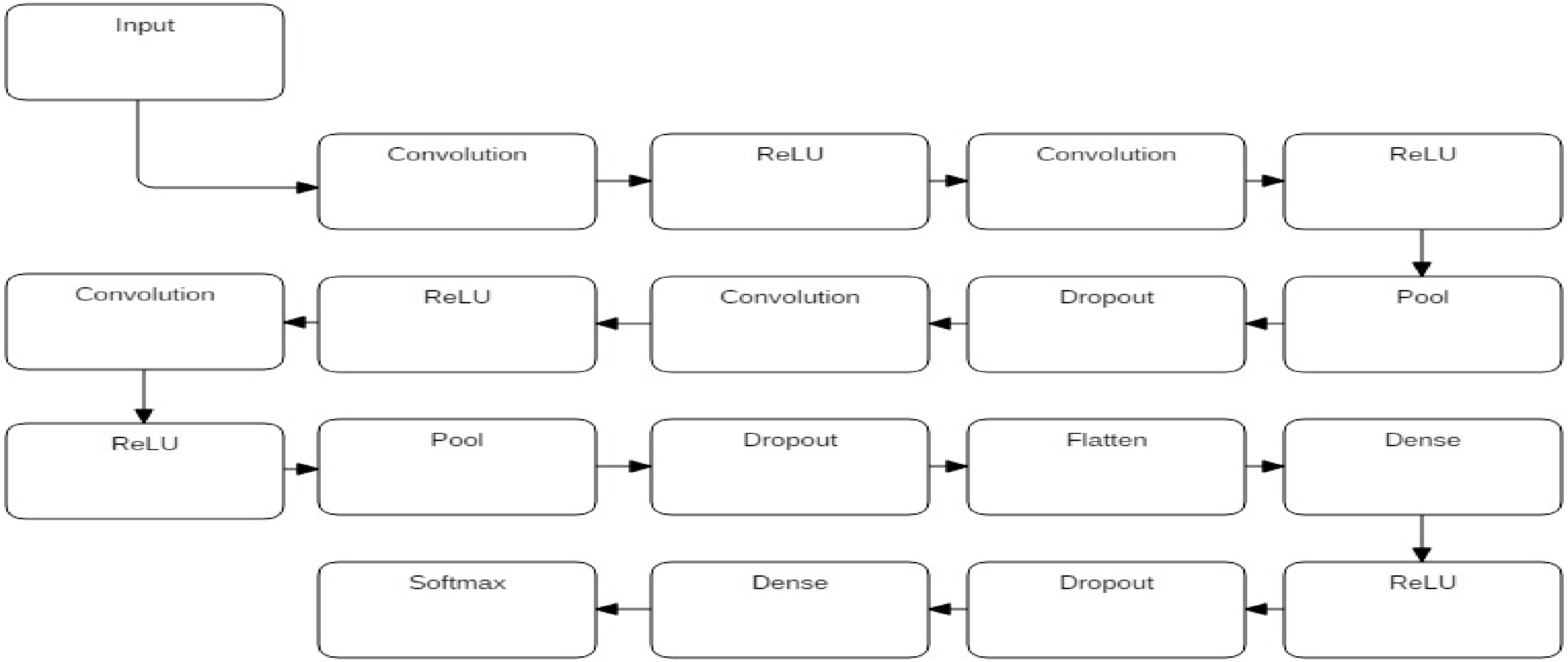


Figure 3.3: Convolution Neural Network Architecture

The most common form of a ConvNet architecture stacks a few CONV-RELU layers, follows them with POOL layers, and repeats this pattern until the image has been merged spatially to a small size. For the purpose of this project, two convolution layers are used, each immediately succeeded by ReLU and the set followed by pooling layer. This pattern is repeated twice and then the data is flattened, densified, ReLU and then the softmax activation is applied to contrain the value assigned to the images such that their sum is 1, i.e. generates a probability vector. In this section the convolutional neural network architecture implemented for this project is discussed in detail

The images undergo pre-processing in which the images are resized to 32X32 pixels. Then the images are normalized by dividing the pixel values by 255 after casting to float values. In the convolution step, a 3X3 filter is passed through an image. It is important to note that filters act as feature detectors from the original input image. The weights in the filter are multiplied with the value in the receptive field, i.e. the pixel value at the location where the filter is passed. Once the multiplication is completed the filter is then shifted according to the stride value. The filter is passed through the entire image. This is where the normalization of data gives an advantage because the pixel multiplication can yield large values but normalization causes these values to be reasonable. It is important to note that, unlike in ILSVRC dataset, normalization improved the computation complexity and time in this network. In the network four convolutional layers are present with the former two using 32 filters and the latter two using 64 filters. The use of smaller filters allowed the use of more number of filters without significant computational cost and allowed better feature extraction evidenced by the improved accuracy in using four convolution layers instead of just two. Adding more convolutional layers improves the accuracy; however, the tradeoff is that the training time increases for a small increases in accuracy. The convolutional layers are used for feature extraction as the multiplication yields the feature map.

The ReLU activation is applied after every convolution to help the network learn non-linear decision boundaries which help to seperate the classes. In effect, ReLU simply eliminates the negative values in the feature map by replacing them with zero. This becomes necessary as an extracted feature map with negative pixel value can cause error in the process. The dimensionality is reduced by using a 2X2 pool matrix. While average and sum pooling are also possible, the best result was obtained via maxpooling which chooses the highest value in 2X2 fields of the feature map.[5] This step ensures that the complexity is minimized. Intuitively, reducing the dimensionality should have significantly affected the performance of the network; however, no such reduction was found when the model was trained and tested as the most important information are still retained. Another reason for using pooling was to minimize overfitting. Overfitting is the condition in which the analysis corresponds too closely to a particular set of data thus hampering reliable future predictions. Overfitting is also resolved by optimizing the parameters used in the model. For example dropout value also affects overfitting. This is discussed later in the section.

The accuracy and the time taken to train the network depends on the learning rate, decay, the number of epochs, and the batch size. Due to memory constraints the batch size used was 128. The number of epochs can be specified with 20 epochs yielding reasonable validation accuracy. Accuracy can be improved by using about 95 epochs. However, currently for testing only 20 epochs has been used and room for improvement exists because the accuracy has not plateaued. The learning rate, decay and other parameters were set using the RMSprop, which is an auto optimizer for recurrent neural networks. For RMSprop, the default learing rate = 0.0001 and decay after each update = 0.0. Deviating from this learning rate caused a sharp decline in performance of the network.[6]

The large number of neurons in the network and their inter-connectivity increases the number of parameters in the model. Thus three dropout layers are used. The function of this layer is to temporarily disconnect the neurons to reduce the parametric complexity of the model. If dropout layers were not introduced in the network, the large number of inter-connections would significantly impact the performance. The dropout value used for this model was 0.25 for training and 0.5 after densifying. Dropout value was 0.25 as large number of disconnection can hamper training. However, towards the end high number of interconnection only increases complexity so dropout value was increased to 0.5. After two blocks of convolution layer, a number of two dimensional feature maps are generated. The flatten layer, in the simplest of words, generates a one dimensional array from these feature maps. The model also has a dense layer (re-connection of neuron for better prediction) with softmax function which is required for multi-class classification problem. The model is trained and class prediction is attempted along with generation of plots of loss and accuracy for both the training and validation sets. The model configuration was saved in .json file format and the training weights were saved in .h5 format to avoid retraining the model for every launch.

3.5 Dataset

The dataset used in the initial phase was self-collected. Due to the lack of sufficient data - only 808 images were used previously - the accuracy of the model suffered significantly. The model tended to overfit the data thus accuracy suffered. To resolve the issue, the Cifar-10 dataset collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton was used.[7]

One problematic aspect of the tiny image dataset is that there are no reliable class labels which makes it hard to use for classification experiments. This issue was resolved in the Cifar-10 dataset. The Cifar-10 dataset consists of 60000 32X32 color images in 10 classes, with 6000 images per class. Furthermore, the dataset is divided into 50000 training images and 10000 test images. The dataset is divided into five training batches and one test batch, each containing 10000 images. The test batch contains exactly 1000 randomly-selected images from each class; the training batch contains the remaining images in random order. The training batches contain exactly 5000 images from each class. The categories in the Cifar-10 dataset include: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The classes are completely mutually exclusive.

CHAPTER 4
EXPECTED OUTPUT

4.1 Work Completed

The Neural Network architecture was designed and trained with around 70% validation accuracy in 20 epochs using Cifar-10 dataset. Previously the data fed into the network was collected by the team and involved 808 images partitioned into four categories: cats, dogs, horses, and humans, where training:validation ratio was 8:2. This yielded a high loss. Increasing the number of images in the dataset improved the validation and testing accuracy within 20 epochs. Improvement is expected as the accuracy curve has not plateaued. The summary of the network was visualized in a table and the plot of accuracy and loss against epochs was visualized for both the training and validation set. The configuration was saved in .json file format and the training weights were saved in .h5 file format to avoid retraining the model after every launch.

4.2 Work Remaining

The following work still remains to be completed:

1. Fine tuning the model to achieve optimum accuracy
2. Design of interactive user interface
3. Error handling procedures remaining
4. Increases images classes from 4 to 10

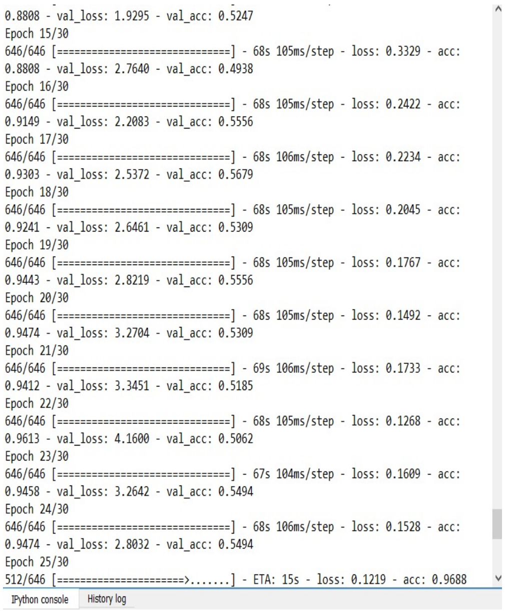
4.3 Problem Faced

Some problems encountered during development were:

- The core concept of the Convolutional Neural Network was difficult to understand
- Some minor glitches occur now and then
- Training time for large dataset is high

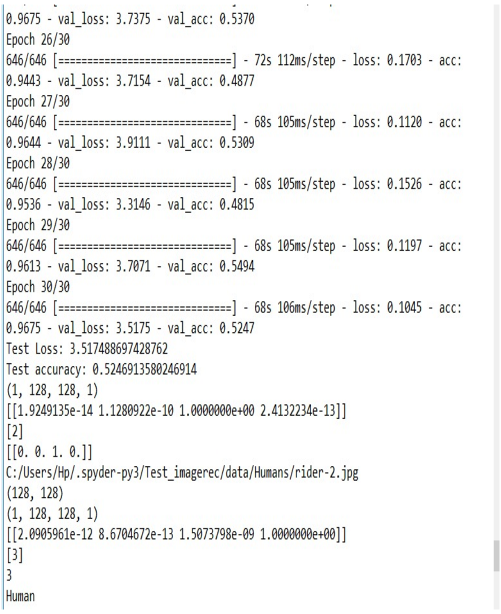
4.4 Output

4.4.1 Previous Output



```
0.8808 - val_loss: 1.9295 - val_acc: 0.5247
Epoch 15/30
646/646 [==============================] - 68s 105ms/step - loss: 0.3329 - acc:
0.8808 - val_loss: 2.7640 - val_acc: 0.4938
Epoch 16/30
646/646 [==============================] - 68s 105ms/step - loss: 0.2422 - acc:
0.9149 - val_loss: 2.2083 - val_acc: 0.5556
Epoch 17/30
646/646 [==============================] - 68s 106ms/step - loss: 0.2234 - acc:
0.9303 - val_loss: 2.5372 - val_acc: 0.5679
Epoch 18/30
646/646 [==============================] - 68s 105ms/step - loss: 0.2045 - acc:
0.9241 - val_loss: 2.6461 - val_acc: 0.5309
Epoch 19/30
646/646 [==============================] - 68s 105ms/step - loss: 0.1767 - acc:
0.9443 - val_loss: 2.8219 - val_acc: 0.5556
Epoch 20/30
646/646 [==============================] - 68s 105ms/step - loss: 0.1492 - acc:
0.9474 - val_loss: 3.2704 - val_acc: 0.5309
Epoch 21/30
646/646 [==============================] - 69s 106ms/step - loss: 0.1733 - acc:
0.9412 - val_loss: 3.3451 - val_acc: 0.5185
Epoch 22/30
646/646 [==============================] - 68s 105ms/step - loss: 0.1268 - acc:
0.9613 - val_loss: 4.1600 - val_acc: 0.5062
Epoch 23/30
646/646 [==============================] - 67s 104ms/step - loss: 0.1609 - acc:
0.9458 - val_loss: 3.2642 - val_acc: 0.5494
Epoch 24/30
646/646 [==============================] - 68s 106ms/step - loss: 0.1528 - acc:
0.9474 - val_loss: 2.8032 - val_acc: 0.5494
Epoch 25/30
512/646 [=======================>.......] - ETA: 15s - loss: 0.1219 - acc: 0.9688
```

Figure 4.1: Training (first build)



```
0.9675 - val_loss: 3.7375 - val_acc: 0.5370
Epoch 26/30
646/646 [==============================] - 72s 112ms/step - loss: 0.1703 - acc:
0.9443 - val_loss: 3.7154 - val_acc: 0.4877
Epoch 27/30
646/646 [==============================] - 68s 105ms/step - loss: 0.1120 - acc:
0.9644 - val_loss: 3.9111 - val_acc: 0.5309
Epoch 28/30
646/646 [==============================] - 68s 105ms/step - loss: 0.1526 - acc:
0.9536 - val_loss: 3.3146 - val_acc: 0.4815
Epoch 29/30
646/646 [==============================] - 68s 105ms/step - loss: 0.1197 - acc:
0.9613 - val_loss: 3.7071 - val_acc: 0.5494
Epoch 30/30
646/646 [==============================] - 68s 106ms/step - loss: 0.1045 - acc:
0.9675 - val_loss: 3.5175 - val_acc: 0.5247
Test Loss: 3.517488697428762
Test accuracy: 0.5246913580246914
(1, 128, 128, 1)
[[1.9249135e-14 1.1280922e-10 1.0000000e+00 2.4132234e-13]]
[2]
[[0. 0. 1. 0.]]
C:/Users/Hp/.spyder-py3/Test_imagerec/data/Humans/rider-2.jpg
(128, 128)
(1, 128, 128, 1)
[[2.0905961e-12 8.6704672e-13 1.5073798e-09 1.0000000e+00]]
[3]
3
Human
```

Figure 4.2: Prediction (first build)

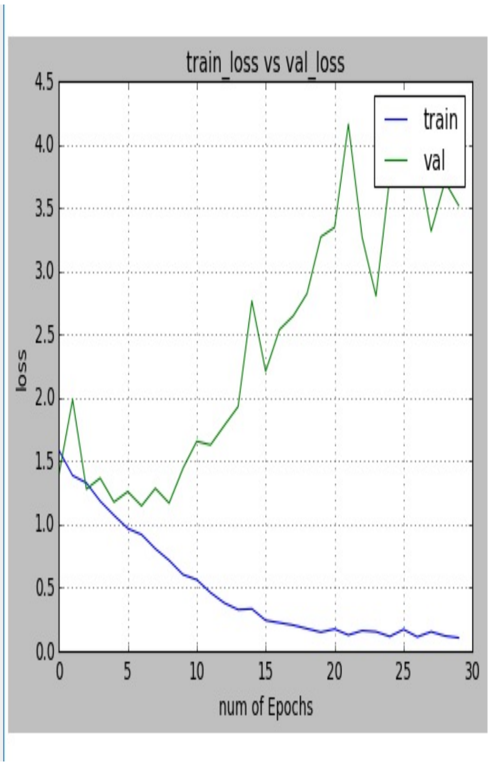


Figure 4.3: Loss Plot (first build)

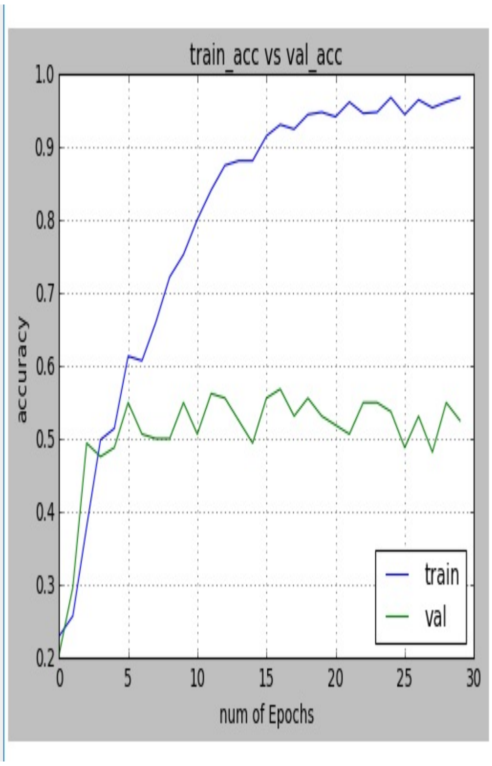


Figure 4.4: Accuracy Plot (first build)

4.4.2 Current Output

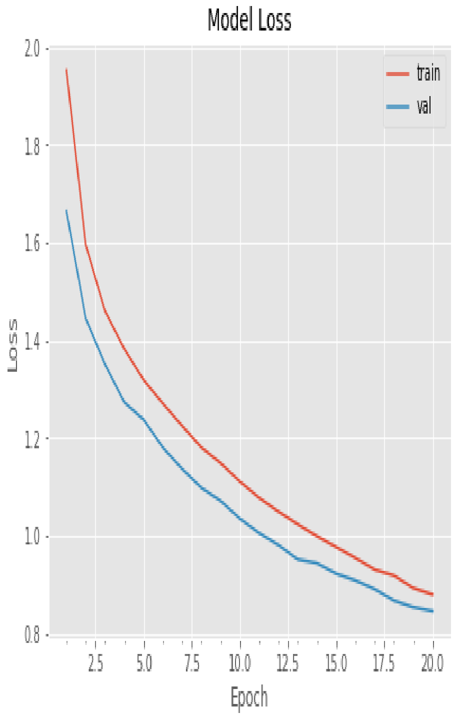


Figure 4.5: Loss Plot

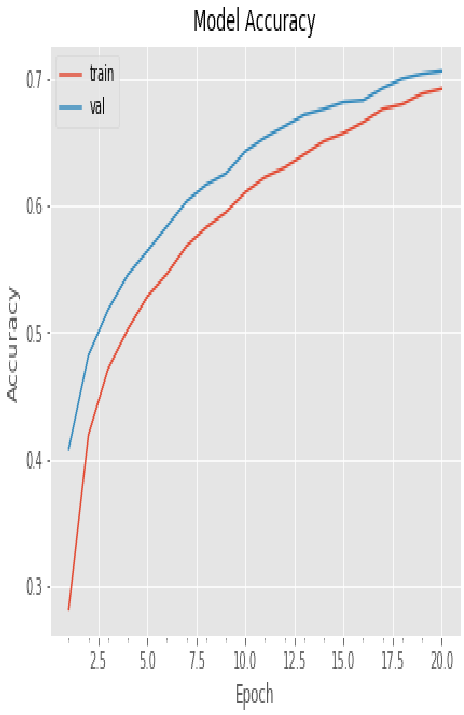


Figure 4.6: Accuracy Plot

4.5 Work Schedule

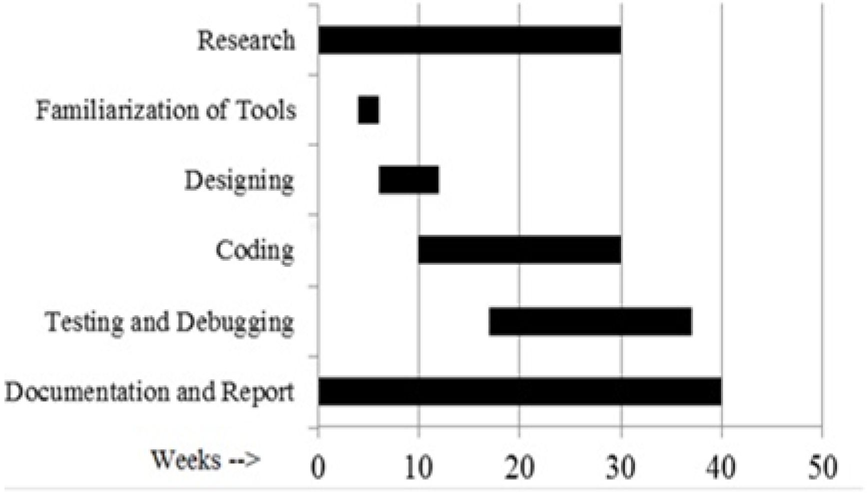


Table 4.1: Gantt Chart

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[2] E. Culurciello, “Neural network architecture,” towardsdatascience.com, 2017. [Online]. Available: https://towardsdatascience.com/neural-network-architectures-156e5bad51ba [Accessed: 26-Feb-2018].

[3] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *CoRR*, vol. abs/1409.1556, 2014. [Online]. Available: http://arxiv.org/abs/1409.1556

[4] I. Sommerville, *Software Engineering*, 9th ed. Boston: Addison-Wesley, 2011. [Online]. Available: https://ifs.host.cs.st-andrews.ac.uk/Books/SE9/

[5] “Cs231n convolutional neural networks for visual recognition,” Cs231n.github.io, 2018. [Online]. Available: https://cs231n.github.io/convolutional-networks/ [Accessed: 03-Mar-2018].

[6] G. Hinton, N. Srivastava, and K. Swersky, “rmsprop: Divide the gradient by a running average of its recent magnitude,” cs.toronto.edu, 2018. [Online]. Available: http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf [Accessed: 01-May-2018].

[7] A. Krizhevsky, “Learning multiple layers of features from tiny images,” cs.toronto.edu, Tech. Rep., 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf [Accessed: 10-Jun-2018].