KANTIPUR ENGINEERING COLLEGE

(Affiliated to Tribhuvan University)

Dhapakhel, Lalitpur

[Subject Code: CT755]

A MAJOR PROJECT FINAL REPORT ON

IMAGE CLASSIFICATION USING CONVOLUTIONAL NEURAL NETWORK

Submitted by:

Aabhushan Pokharel	**[44042]**
Prabin Bhusal	**[44057]**
Rajesh Mahato	**[44061]**

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

August, 2018

IMAGE CLASSIFICATION USING CONVOLUTIONAL NEURAL NETWORK

Submitted by:

Aabhushan Pokharel [44042]

Prabin Bhusal [44057]

Rajesh Mahato [44061]

Supervised by:

Sudhir Shakya

Founder

Sorus Nepal Pvt. Ltd.

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

August, 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 Network are special neural networks that are suited to this task. This report corresponds to the implementation 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, generated by applying softmax activation. The accuracy of the CNN for this project was concluded at 78% for 10 classes on the Cifar-10 dataset. The model of the network was trained and tested on standard laptop CPUs. This document includes a step-by-step walkthrough of the project and details the intricacies of the system under implementation.

Keywords – Image Classification, Convolutional Neural Network

ACKNOWLEDGMENT

It gives us immense pleasure to express our deepest sense of gratitude and sincere thanks to our highly respected and esteemed guide Mr. Sudhir Shakya, Founder, Sorus Nepal Pvt. Ltd., for his valuable guidance, encouragement and help for completing this work. His useful suggestions for this whole work and co-operative behavior are sincerely acknowledged.

We would like to express our sincere thank to Er. Rabindra Khati, Head of Department, Department of Electronics and Computer Engineering, Kantipur Engineering College, for his whole hearted support.

We are also grateful to all our teachers for their constant support and guidance.

At the end we would like to express our sincere thanks to all our friends and others who helped us directly or indirectly during this project work.

Aabhushan Pokharel	[44042]
Prabin Bhusal	[44057]
Rajesh Mahato	[44061]

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

CNN Convolutional Neural Network

ReLU Rectified Linearity Unit

CHAPTER 1
INTRODUCTION

1.1 Background

The amount of visual data present in the world is expanding exponentially every day. With advances in photographic technology and easy access to smartphones, this is unlikely to slow any time soon. This has created a problem of identifying the content in an image, and many companies are investing billions of dollars to develop a reliable visual recognition system implementing different methods.

Convolutional Neural Network provides a way to classify these images using an architecture and process that is more transparent that the traditional neural networks. A Convolutional Neural Network is a specific type of artificial neural network that uses perceptrons, a machine learning unit algorithm, for supervised learning to analyze data. CNNs apply to image processing, natural language processing and other kinds of cognitive tasks. Like other kinds of artificial neural networks, a Convolutional Neural Network has an input layer, an output layer and various hidden layers. Some of these layers are convolutional, using a mathematical model to pass on results to successive layers. This simulates some of the actions in the human visual cortex. CNNs are a fundamental example of deep learning, where a more sophisticated model pushes the evolution of artificial intelligence by offering systems that simulate different types of biological human brain activity.

A Convolutional Neural Network is also referred to as ConvNet.

While traditional Neural Networks provided a methodology for image classification, the accuracy left for improvement. Also the scientific community was not entirely comfortable with the results of an algorithm that was not understandable by humans. Convolutional Neural Network was a solution to that. It was found that the CNN improved accuracy of classification when used with large datasets. Also, the workings of Convolutional Neural Network was understandable in the sense that, in theory, it was possible to work out the results of the convolutional layer in the neural network. The major

layers in the network are the conolutional layers, Rectified Linearity Unit (ReLU) and pooling with additional layers such as dense, flatten and softmax activation added for various reasons.

The output of the CNN is a list of probabilities assigned to each class. The CNN classifies the image according to the highest assigned probabilities also called confidence levels. So, if an image contains a dog, a CNN would assign a high confidence level of 0.94 to the dog while only 0.06 to a cat in a two class training dataset. The confidence levels are assigned according to the weights which are adjusted by the CNN once 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
- 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 $a^i_{x,y}$ the activity of a neuron computed by applying kernel *i* at position *(x,y)* and then applying the ReLU nonlinearity, the response normalized activity $b^i_{x,y}$ is given by the expression

$b^i_{(x,y)} = a^i_{(x,y)}/(k+\alpha \sum_{j=max(0,i-n/2)}^{min(N-1,i+n/2)} (a^j_{(x,y)})^2)^\beta$ 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 CNN 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
RELATED THEORY

3.1 Convolutional Neural Network

Convolutional Neural Network is a class of deep, feed forward artificial neural networks, most commonly applied to analyzing visual imagery. Convolutional Neural Networks were inspired by biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. CNNs are preferred because they require very little pre-processing compared to traditional neural networks. The network learns filters which in other neural networks were engineered. This provides CNNs an advantage over the traditional neural networks. A CNN contains the following layers:

- Convolutional layer
- Max-Pooling
- Fully-Connected layer

The layers are discussed further below.

3.1.1 Convolutional Layer

Convolutional layers consist of a rectangular grid of neurons. It requires that the previous layer also be a rectangular grid of neurons. Each neuron takes inputs from a rectangular section of the previous layer; the weights for this rectangular section are the same for each neuron in the convolutional layer. Thus, the convolutional layer is just an image convolution of the previous layer, where the weights specify the convolution filter.

The feature maps are extracted via the dot product multiplication between the filter value and the receptive field in the input to the convolutional layer. In addition, there may be several grids in each convolutional layer; each grid takes inputs from all the grids in the previous layer, using potentially different filters.

The output dimensions of the convolutional layers are calculated as follows:

$$\boldsymbol{outputwidth} = \frac{\boldsymbol{W - F_w + 2P}}{\boldsymbol{S_w}} + \mathbf{1} \text{ and } \boldsymbol{outputheight} = \frac{\boldsymbol{H - F_h + 2P}}{\boldsymbol{S_h}} + \mathbf{1}$$

where W = width, H = height, F_w = width of filter, F_h = height of filter, and S = Stride. [4]

3.1.2 Max-Pooling

After each convolutional layer, there may be a pooling layer. The pooling layer takes small rectangular blocks from the convolutional layer and subsamples it to produce a single output from that block. There are several ways to do this pooling, such as taking the average or the maximum, or a learned linear combination of the neurons in the block. The best result is given by the max-pooling layers; that is, they take the maximum of the block they are pooling.

3.1.3 ReLU

Instead of sigmoids, most recent deep learning networks use Rectified Linearity Units (ReLUs) for the hidden layers. A ReLU has output 0 if the input is less than 0, and raw output otherwise. That is, if the input is greater than 0, the output is equal to the input. ReLUs' machinery is more like a real neuron in your body.

$$\boldsymbol{f(x) = max(x, 0)}$$

ReLU activations are the simplest non-linear activation function can be used. When the input is positive, the derivative is just 1, so there isn't the squeezing effect that is met on backpropagated errors from the sigmoid function. Research has shown that ReLUs result in much faster training for large networks.

3.1.4 Softmax Activation

The softmax function squashes the outputs of each unit to be between 0 and 1, just like a sigmoid function. But it also divides each output such that the total sum of the outputs is equal to 1. The output of the softmax function is equivalent to a categorical probability distribution, it tells you the probability that any of the classes are true.

Mathematically the softmax function is shown below, where z is a vector of the inputs to the output layer (if there are 10 output units, then there are 10 elements in z). And again, j indexes the output units, so j = 1, 2, ..., K.

$$\sigma(z)_j = \frac{e^{Z_j}}{\sum_{K}^{k=1} e^{Zk}}$$ [5]

CHAPTER 4
METHODOLOGY

4.1 Software Development Model

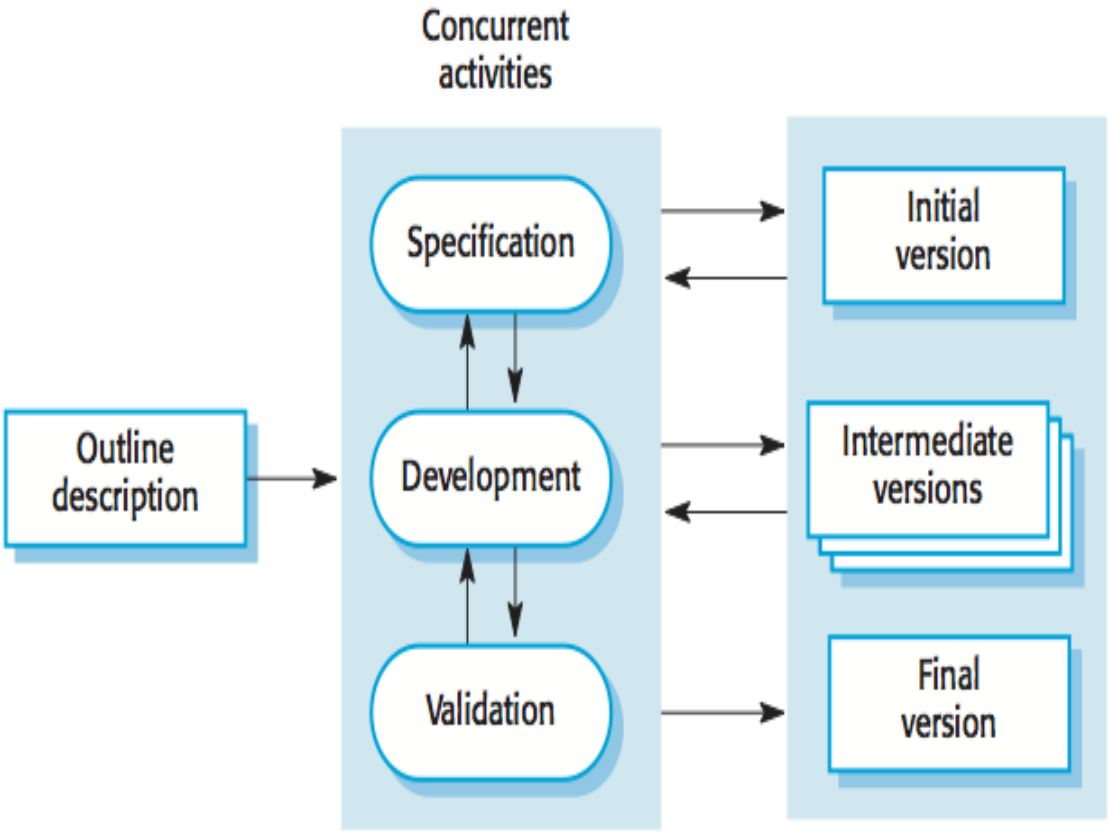

Figure 4.1: Incremental Model of Software Development[6]

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

4.2 Installing the Development Environment

The project was implemented using Anaconda distribution. The following commands were entered in the anaconda prompt.

1. Install tensorflow backend
   (a) Create conda environment named tensorflow by invoking the following command: **conda create -n tensorflow pip python=3.5**
   (b) Activate conda environment : **activate tensorflow**
   (c) Install : **pip install - -ignore-installed - -upgrade tensorflow**
2. Install Keras library: **conda install keras**

4.3 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

4.4 Flowchart

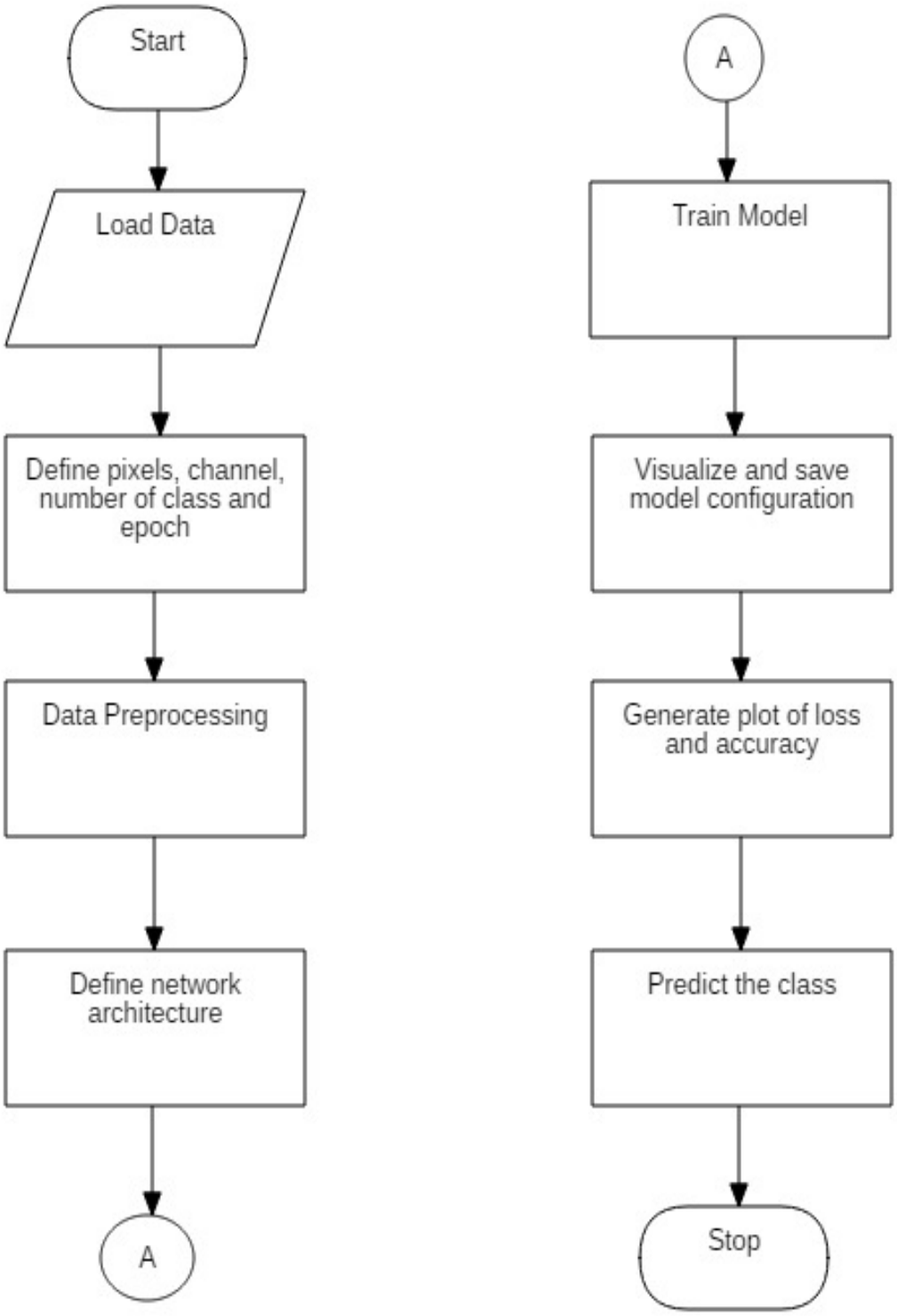

Figure 4.2: Flowchart for the process

4.5 Model Diagram

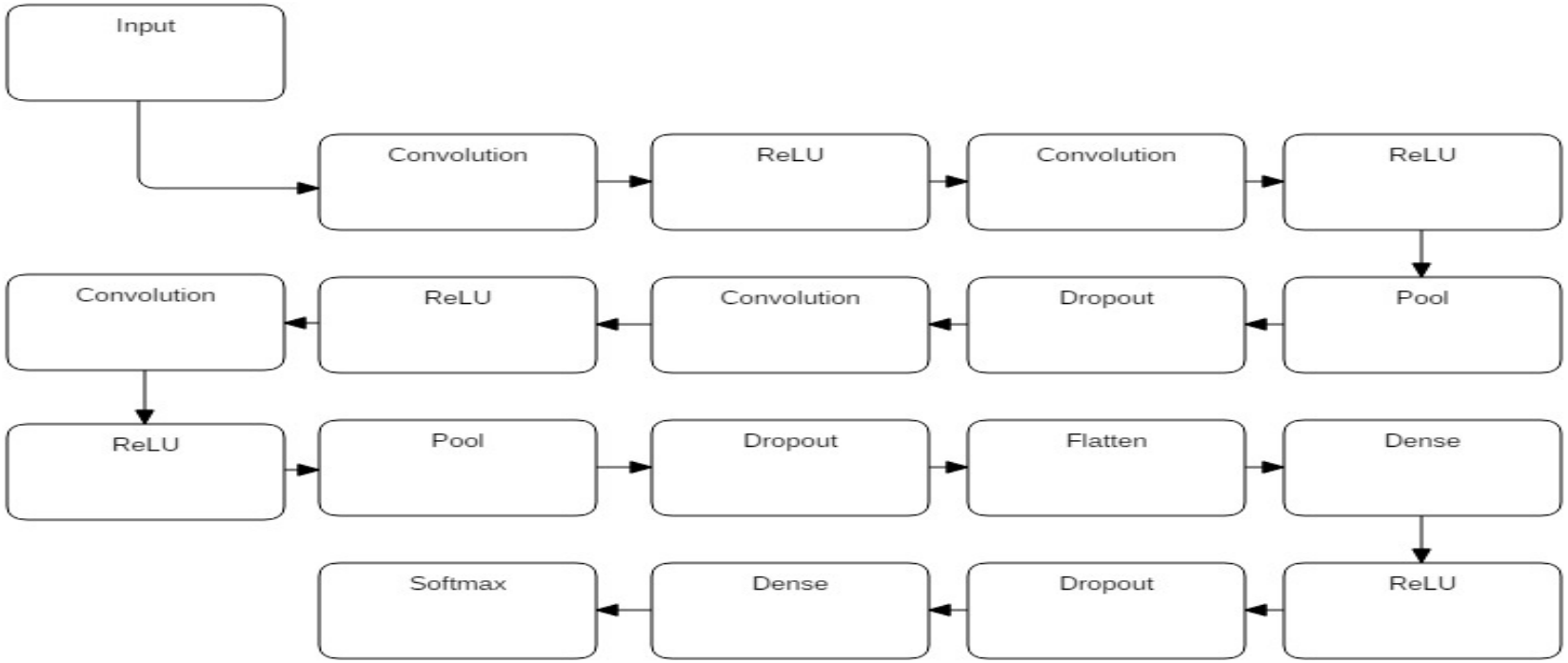

Figure 4.3: Model Diagram

4.6 Model Implementation

The model displayed in the preceding page is implemented via the following code:

```
def create_network(channels, image_rows, image_cols, lr,
    decay, momentum):
    model = Sequential()

    model.add(Conv2D(32, (3, 3), padding='valid', input_shape =
        (image_rows, image_cols, channels)))
    model.add(Activation('relu'))
    model.add(Conv2D(32, (3, 3)))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Conv2D(64, (3, 3), padding='valid'))
    model.add(Activation('relu'))
    model.add(Conv2D(64, (3, 3)))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Flatten())
    model.add(Dense(512))
    model.add(Activation('relu'))
    model.add(Dropout(0.5))
    model.add(Dense(nb_classes))
    model.add(Activation('softmax'))

    sgd = SGD(lr=lr, decay=decay, momentum=momentum,
        nesterov=True)

    model.compile(loss='categorical_crossentropy',
        optimizer=sgd)
```

```
    model.summary()
    return model

model = create_network(3, 32, 32, 0.01, 1e-6, 0.9)
```

The input and outputs of each layer are described below:

1. Convolution
    - takes 32x32 image as input
    - uses 32 2x2 filters(randomly generated) to slide over the image and computes dot product between filter value and the receptive field
    - outputs 30x30 feature map of depth 32 (see 3.1.1)
2. ReLU
    - applies $\boldsymbol{f(x) = max(x, 0)}$ to each feature map obtained from previous layer
3. Convolution
    - takes output from previous layer as input
    - uses 32 2x2 filters to generate feature map
    - outputs 28x28 feature map of depth 32
4. Rectified Linearity Unit
    - applies $\boldsymbol{f(x) = max(x, 0)}$ to each feature map obtained from previous layer
5. Max-Pooling
    - Pool matrix size is 2x2
    - chooses the highest value in the 2x2 field
    - output dimension is 14x14
6. Dropout
    - rate = 0.25, i.e neurons are dropped with probability of 25%
7. Convolution
    - input dimension is 14x14
    - uses 64 2x2 filters (discussed in the next chapter)
    - outputs 12x12 feature map of depth 64

8. ReLU
    - applies $f(x) = max(x, 0)$ to each feature map obtained from previous layer
9. Convolution
    - takes output from previous layer as input
    - uses 32 2x2 filters to generate feature map
    - output dimension is 10x10 with depth 64
10. Rectified Linearity Unit
    - applies $f(x) = max(x, 0)$ to each feature map obtained from previous layer
11. Max-Pooling
    - 2x2 pool matrix
    - output dimensions 5x5 with depth 64
12. Dropout
    - rate = 0.25, i.e neurons are dropped with probability of 25%
    - helps to reduce overfitting
13. Flatten
    - Flattens the 5x5x64 output from previous layers into a single vector array
14. Dense, Dropout, Dense
    - Perform initial regular training, but with the main purpose of seeing which weights are important, not learning the final weight values
    - Drop the connections where the weights are under a particular threshold
    - Make the network dense again and retrain it
15. Softmax
    - generates list of probabilities (see 3.1.4)

This is summarized neatly by Table 4.1:

The code snippet shown in the next page trains the model described by the code snippet preceding that. The first two parameters in model.fit() represent the training and validation sets and the following two represent the number of epochs and the batch size. Checkpoint and earlystop are used to stop the training process if the loss does not decrease in the three successive steps.

Table 4.1: Model Summary

Layer (type)	Output Shape	Param #
conv2d_5 (Conv2D)	(None, 30, 30, 32)	896
activation_7 (Activation)	(None, 30, 30, 32)	0
conv2d_6 (Conv2D)	(None, 28, 28, 32)	9248
activation_8 (Activation)	(None, 28, 28, 32)	0
max_pooling2d_3 (MaxPooling2	(None, 14, 14, 32)	0
dropout_4 (Dropout)	(None, 14, 14, 32)	0
conv2d_7 (Conv2D)	(None, 12, 12, 64)	18496
activation_9 (Activation)	(None, 12, 12, 64)	0
conv2d_8 (Conv2D)	(None, 10, 10, 64)	36928
activation_10 (Activation)	(None, 10, 10, 64)	0
max_pooling2d_4 (MaxPooling2	(None, 5, 5, 64)	0
dropout_5 (Dropout)	(None, 5, 5, 64)	0
flatten_2 (Flatten)	(None, 1600)	0
dense_3 (Dense)	(None, 512)	819712
activation_11 (Activation)	(None, 512)	0
dropout_6 (Dropout)	(None, 512)	0
dense_4 (Dense)	(None, 10)	5130
activation_12 (Activation)	(None, 10)	0

Total params: 890,410
Trainable params: 890,410
Non-trainable params: 0

The model configuration was stored in .json file format and the training weights are stored in .h5 file format.

```
earlystop = EarlyStopping(monitor='val_loss', patience=3,
   verbose=1, mode='auto')
checkpoint = ModelCheckpoint(filepath =
   'best_model_0718.hdf5', verbose = 1, save_best_only = True)

model.fit(X_train, y_train, nb_epoch=30, batch_size=128,
         validation_split=0.1, verbose=1,
            callbacks=[checkpoint, earlystop])
```

4.7 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.

In the final build, the Cifar-10 dataset was imported from the Keras library itself instead of importing it manually for convenience.

4.8 Version Summary

Table 4.2: Version Summary

Version	Features	Comment
Build 1	Self-collected dataset	54% accuracy due to small dataset and unoptimized model
	Contains 808 images split in 4:1 test:validate ratio	
	Classification upto four classes	
	128x128 pixel images pre-processed	
	Images were converted to grayscale	
Build 2	Cifar-10 dataset	69% validation accuracy achieved; however, the model seemed to overfit data so misclassified most raw data
	Train:test split in 5:1 ratio	
	Dataset unpickled via own code	
	Classification upto 10 classes	
	32x32 pixel images as specified in dataset	
	Color images so no grayscale conversion necessary	
Build 3	Cifar-10 dataset imported from keras	78% accuracy achieved and can classify most raw data accurately
	Train:test split in 5:1 ratio	
	Model optimization by tweaking parameters	
	Classification upto 10 classes	

CHAPTER 5
RESULT AND DISCUSSION

5.1 Previous Output

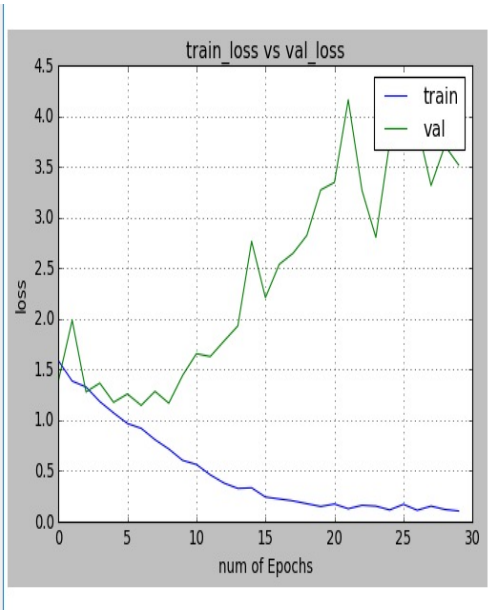

Figure 5.1: Loss Plot (build 1)

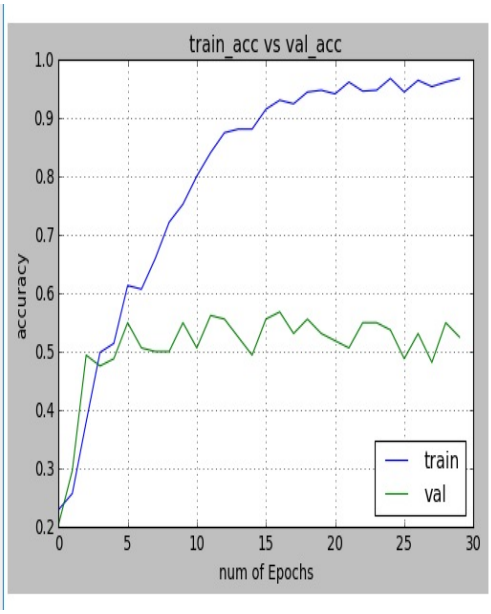

Figure 5.2: Accuracy Plot (build 1)

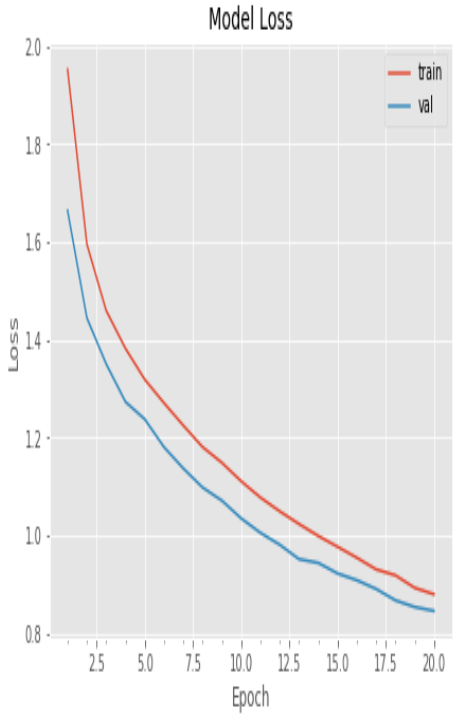

Figure 5.3: Loss Plot (build 2)

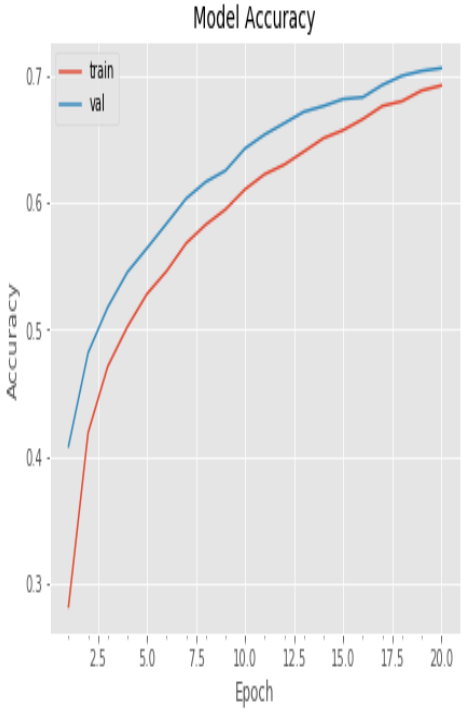

Figure 5.4: Accuracy Plot (build 2)

5.2 Current Output

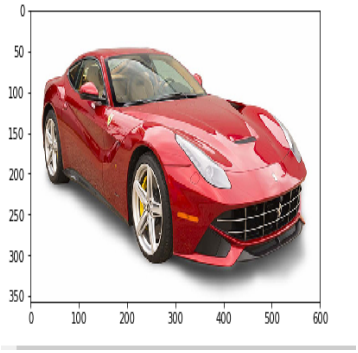

Figure 5.5: Automobile

```
Predicted Class: automobile
Predicted Probabilities:
airplane : 1.0623273e-07
automobile : 0.9996861
bird : 7.287953e-10
cat : 2.7694349e-11
deer : 3.6572304e-11
dog : 4.726108e-10
frog : 5.1707183e-08
horse : 1.8440745e-11
ship : 1.7099454e-08
truck : 0.00031377253
```

Figure 5.6: Probability List

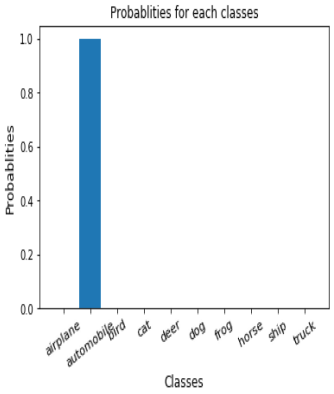

Figure 5.7: Bar Plot for Automobile

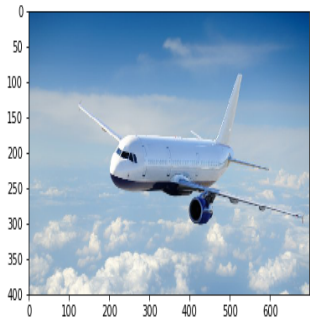

Figure 5.8: Airplane

```
Predicted Probabilities:
airplane : 0.5765929
automobile : 0.00066861545
bird : 0.28963673
cat : 0.0060407114
deer : 0.03541719
dog : 0.0011625584
frog : 0.0040938333
horse : 0.0009568596
ship : 0.08525898
truck : 0.00017154639
```

Figure 5.9: Probability List

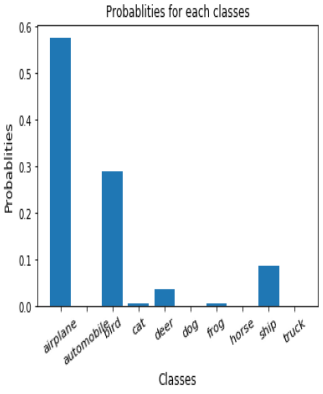

Figure 5.10: Bar Plot for Airplane

- Build 1 yielded high loss and low accuracy
- Data used was self collected, and model was not tuned
- Build 2 performed significantly better in terms of loss and accuracy(training)
- The data was overfitting, as misclassification rate was high
- Build 3 yielded good accuracy and low loss
- System was able to classify the images correctly as shown in Figures 5.5 through 5.10.

5.3 Discussion

Table 5.1: Parametric Summary

Parameter	Value
number of epochs	30
learning rate	0.01
decay	10^{-6}
momentum	0.9
batch size	128
filter number	32, 32, 64, 64
dropout value	0.25, 0.5
number of classes	10
Train:test ratio	5:1

Based on the above hyper-parameters, the training time for the network, for 30 epochs, was 163 minutes in an Intel(R) Core(TM) i7-7500U CPU @ 2.70GHZ 2.90GHZ processor with 8.00 GB RAM and the accuracy obtained was approximately 78%. The number of epochs was determined by using the earlystop feature discussed in section 4.6. The learning rate was set as lr = 0.01. Lowering the learning rate meant that each iteration took longer to finish, i.e convergence became very slow; this led to overfitting of the training set, thus decreased accuracy. Increasing the rate above 0.01 led to large jumps in learning which caused divergence. The decay value was set as 10^{-6}. This value was set according to the value provided in the Keras documentation, as was the momentum. The reason for decay value being low was to prevent the learning rate to decrease rapidly to zero and preventing the network from converging. The batch size was chosen as 128 due to memory constraints in a standard CPU. A larger batch was possible to use; however, doing so caused higher memory usage yet no change in accuracy. The number of filters used was 32 for the first two convolution steps and 64 for the remaining two. At the end of the first two convolution steps, max-pooling was done and (as shown in 4.1) the output dimension was halved. This meant that there was less features to extract from the image at this stage so the number of filters needed to be doubled to 64 to offset this change. When this was not set as such, as in the first two builds, the accuracy dropped by varying degrees. The dropout value initially was

0.25, i.e 25% of the neurons were disconnected, to reduce computational complexity. Neurons with weights below a certain threshold were disconnected to reduce the noise in the network. This was later increased to 50% for the same reason. Also the number of classes in the first build was four due to only 808 images being present (split in 4:1 ratio) and included the classes: cat, dog, horses, and humans. Upgrade to 10 classes was made when the Cifar-10 dataset was successfully integrated. Other parameters were set by using the sgd (Stochastic Gradient Decay) optimizer due to it being the standard optimizer being used in such systems. For the split in Cifar-10 dataset see 4.7.

5.4 Limitations

1. The model needs to be retrained from scratch if new data are to be added to the training dataset
2. Can only classify mutually exclusive images

5.5 Work Schedule

Table 5.2: Work Schedule

Parameter	Start Date	End Date	Duration (Days)
Research	20-Dec-17	01-Aug-18	224
Familiarization with tools	24-Jan-18	20-Feb-18	28
Build 1	26-Feb-18	14-May-18	78
-Designing	26-Feb-18	14-Mar-18	17
-Coding	13-Apr-18	11-May-18	29
-Testing	07-May-18	14-May-18	8
Build 2	22-May-18	10-Jul-18	50
-Designing	22-May-18	08-Jun-18	18
-Coding	04-Jun-18	06-Jul-18	33
-Testing	21-Jun-18	10-Jul-18	20
Build 3	11-Jul-18	09-Aug-18	30
-Designing	11-Jul-18	18-Jul-18	8
-Coding	16-Jul-18	06-Aug-18	22
-Testing	30-Jul-18	09-Aug-18	11
Documentation	01-Jan-18	09-Aug-18	221

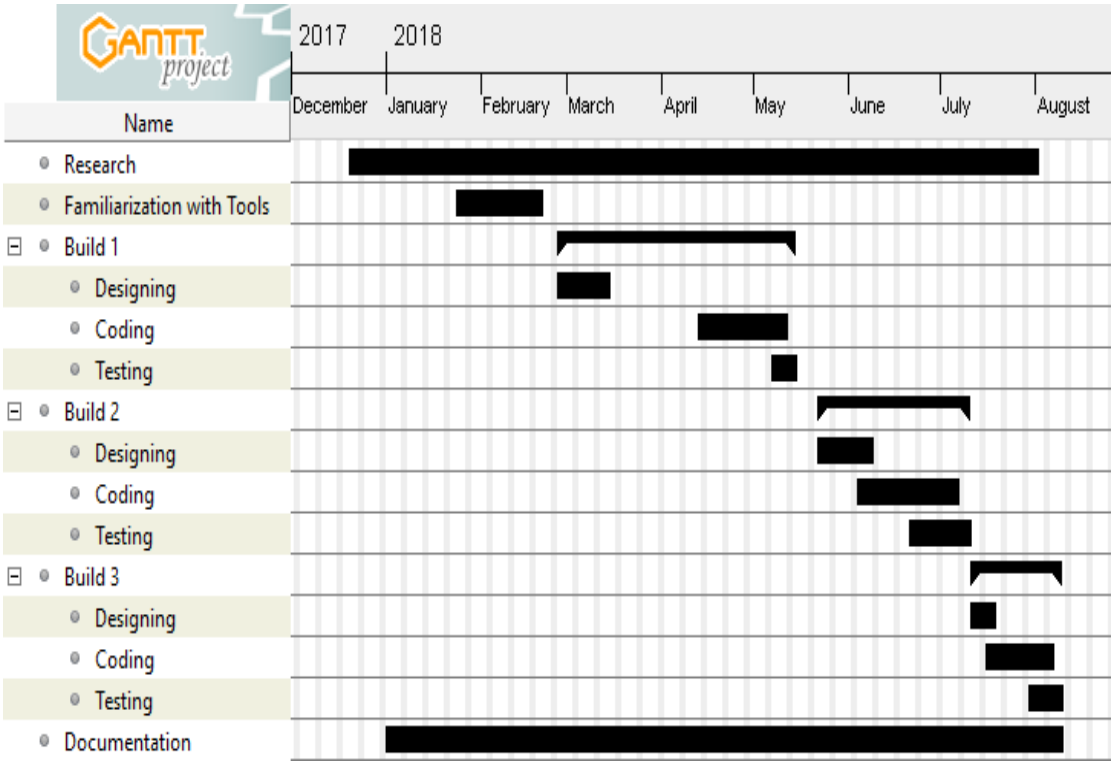

Figure 5.11: Gantt Chart

CHAPTER 6
CONCLUSION AND FUTURE ENHANCEMENT

This project was primarily concerned with the implementation of Convolutional Neural Network to develop an image classification system. Despite the availability of different methods to accomplish such classification tasks, advances in deep learning has enabled CNNs to be effective networks for this task. A well implemented Convolutional Neural Network is capable of classifying images with high accuracy.

The Convolutional Neural Network model implemented in this project utilized multiple convolutional layers, each followed by a Rectified Linearity Unit layer, and each pair of CNN was followed by a max-pool layer. A dense-dropout-dense sequence was used to improve the performance of the model based on previous work done. A softmax activation function was used to generate a list of probabilities for each class that sum to 1 based on which the classifications were made.

The project was completed in three stages using the incremental software development life cycle in python programming language. The first build included a dataset of 808 images split in 4:1 train:test ratio and classified upto four classes. The accuracy of the build was approximately 54% and peaked at 5 epochs. The second build used the Cifar-10 dataset in the prescribed format; due to the parameters not being fine tuned, this version misclassified the images despite having around 69% validation accuracy, showing characteristics of overfitting. This build included 10 classes. The final build was successfully developed with approximately 78% accuracy and peaked at 30 epoch. This build dealt with the issues encountered in the previous build.

Based on the experience of working on this project, the following enhancements could be made to the system:

- Further tune the model to push the accuracy above 85%
- Increase in the number of classification classes with increase in number of images in dataset

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

[5] "Relu and softmax function," github.com, 2017. [Online]. Available: https://github.com/Kulbear/deep-learning-nano-foundation/wiki/ReLU-and-Softmax-Activation-Functions [Accessed: 20-May-2018].

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

[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].