Ahsanullah University of Science and Technology
Department of Computer Science and Engineering
Spring 21

Soft Computing Lab
CSE 4238

Report on Assignment - 1

Implementation of Neural Networks and Convolutional Neural Networks on a Given Dataset

---

Submitted by

Abrar Rafid Noor
ID: 170204059
Group: B1

Submitted to

Mr. H M Zabir Haque
Assistant Professor

Mr. Nibir Chandra Mandal
Lecturer

Task: Implementing Neural Networks (with specified hyper-parameters) and Convolutional Neural Networks on a given dataset and try to show that the CNN model performs 50% better than that of the NN model.

Dataset: For the given task, we have been provided *two folders* for the use of the dataset. They are:

- [ ] Firmware
- [ ] Imagery

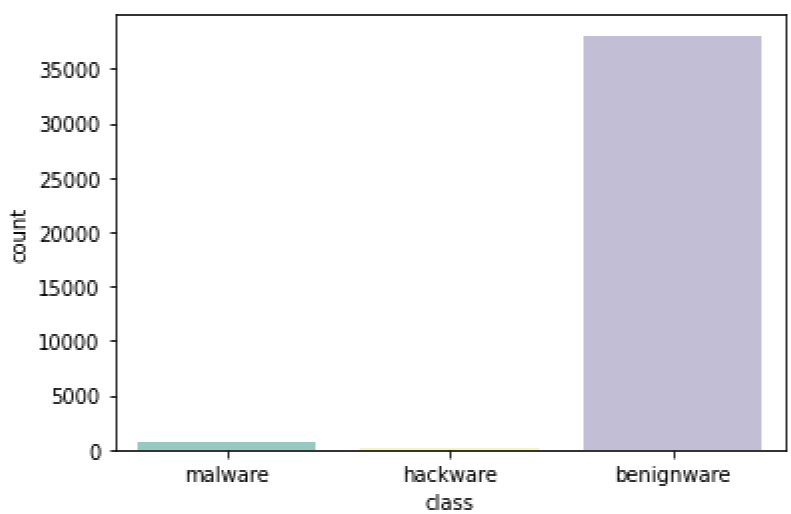
Firmware contains *firmware.csv* file which have columns named filename (*names of the image files*), class (*their class name*), target (*numerical representation of the class*) and additional 1024 columns containing the pixel values (0 *to* 255) for each of the 1024 pixels.

It has been told to use only the *filename* and *class* column and drop off the rest of the columns, for our task.

Pre processing: In the given csv file, three classes were present. They were:

- [ ] benignware
- [ ] malware
- [ ] hackware

First, let's see the number of file names given per class.



We can see, in the *firmware.csv* file, almost 98% of the data are given of *benignware* class.

At the same time, total image files given in the *imagery* folder is 4482. In which 2999 benignware class images are provided. So, we can presume *undersampling* has been applied for the image dataset (although more than 66% images are still benignware). Also classes of image provided were *benignware*, *malware*, *hackware* and *gray*. Gray class image file names were not provided in the csv file.
In the next step, we will create a modified csv file that will only contain the filenames of the provided images with the help of the existing *firmware.csv* file.
And then, add the names of the gray image files into the new csv file.
Hence it will have

- ☐ 2999 *benignware* class labeled filenames
- ☐ 711 *malware* class labeled filenames
- ☐ 699 *gray* class labeled filenames
- ☐ 103 *hackware* class labeled filenames

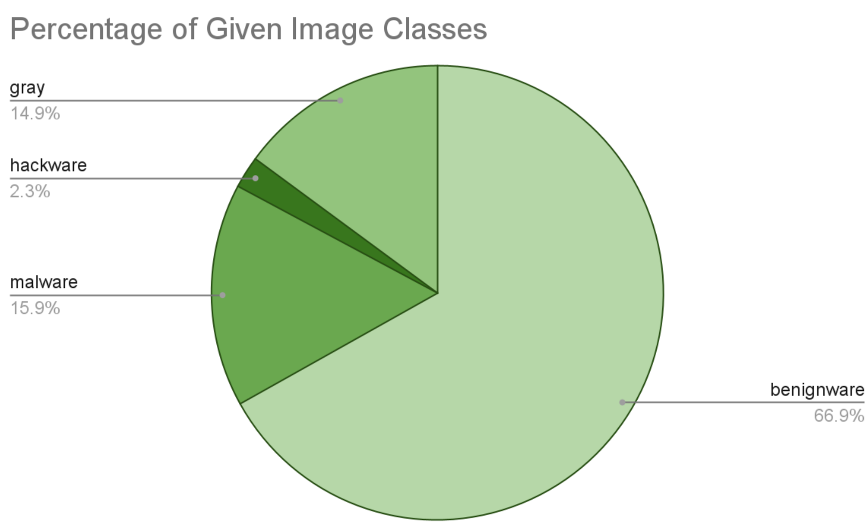


Fig: Pie chart of the class samples in the dataset

The csv file will be used to create our dataset and later be used in the *data loader* to create our *train loader* and *test loader*.

In the following, training setups and their results for both the neural networks and convolutional neural networks models is described.

Setups and Results for Neural Networks Model

Hyper-parameters used for training the model:

Number of hidden layers: 5
Number of nodes in hidden layers: 100
Number of iterations: 2000
Learning rate: 0.001
Batch size: 20

Optimizer: Adam
Number of Epochs: 12 (*calculated with given iteration, batch size and length of the dataset*)
Loss function: Cross Entropy
Activation function: ReLU

Model Architecture:

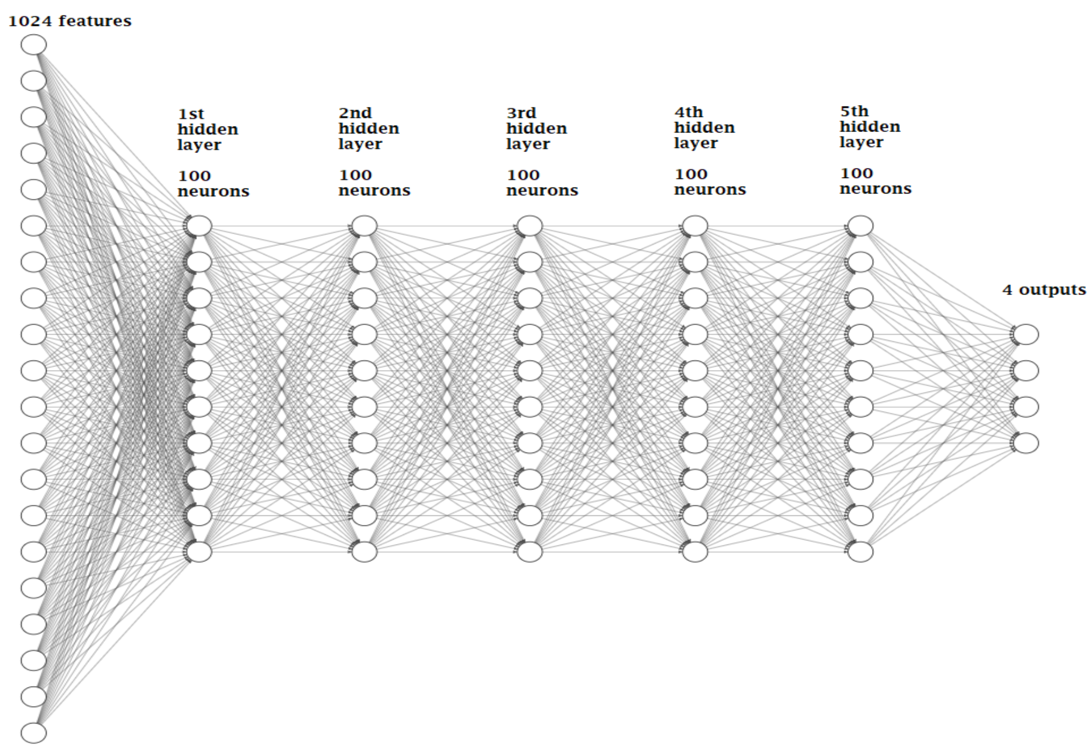
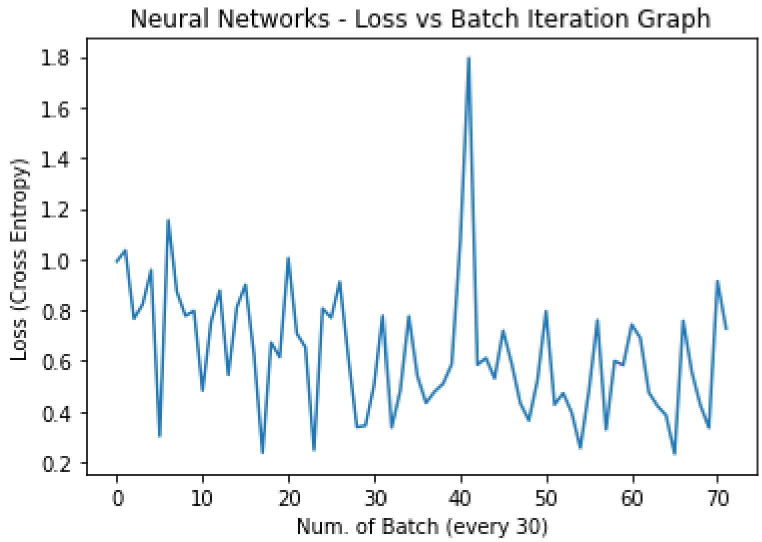


Fig: Neural Networks model architecture (abstract)
Model has been generated using http://alexlenail.me/NN-SVG/index.html

Training:

Total training sample size: 3586

The loss vs batch iteration graph for training the NN is given below:



Testing:

Total testing sample size: 896

Class	benignware	malware	hackware	gray
Samples	595	144	23	134
Predicted	763	0	0	133
Correct	592	0	0	130

Results Obtained: *F-1 Score:* 0.46329673738675287

Result Analysis:

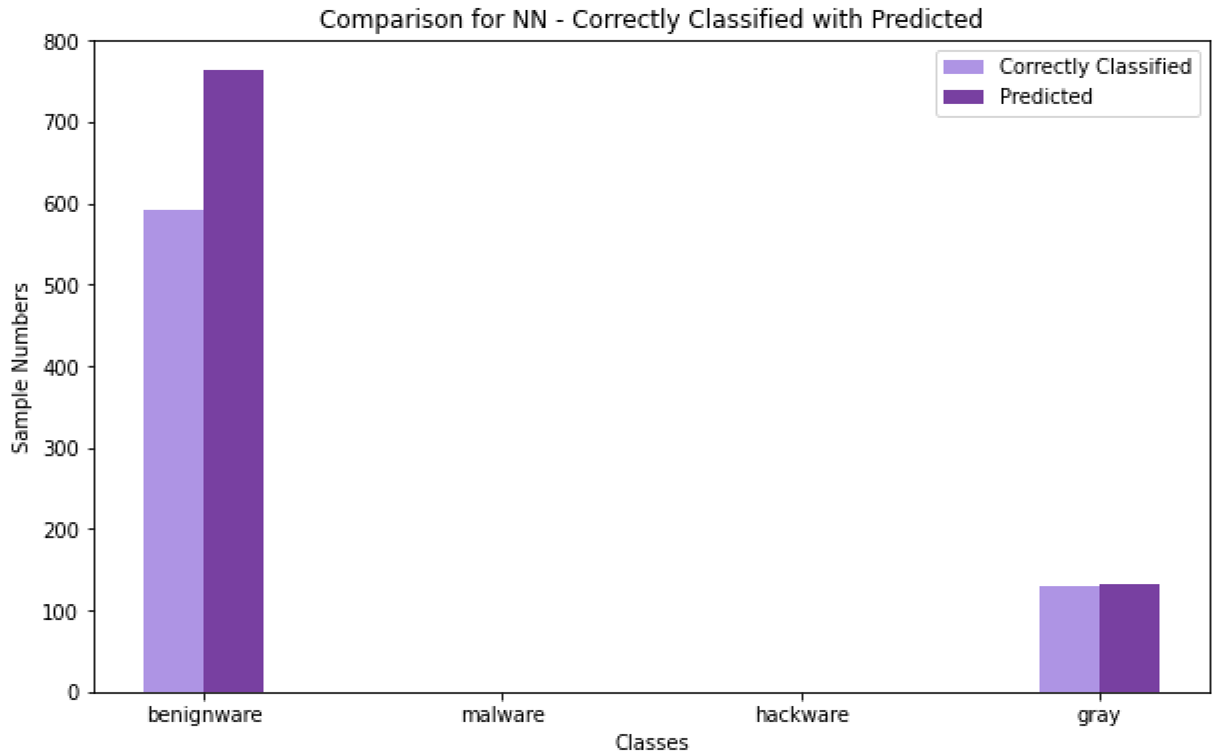


Fig: Bar diagram of classes correctly classified with their sample size

Our neural networks model could only learn the classes of *benignware* and *gray*. One of the reasons may be- the number of *benignware* class labeled samples were higher than the other classes. And, for *gray* images, perhaps it was easier to learn and classify than *malware* and *hackware*, as all the pixel values are the same for input *gray* class images.

Objective for Convolutional Neural Networks Model:

As per our assignment requirement, F-1 Score of the Convolutional Neural Networks model should be **0.69495** (50% *better than* 0.4633) or above.

Setups and Results for Convolutional Neural Networks Models

Hyper-parameters used for training the model:

Number of iterations: 2000
Learning rate: 0.001
Batch size: 20
Optimizer: Adam
Number of Epochs: 12 (*calculated with given iteration, batch size and length of the dataset*)
Loss function: Cross Entropy
Filter size: 3x3
Convolutional layers: 3
Fully-connected layers: 4
Pooling: Maxpool (2x2)
Activation function: ReLU

Model Architecture:

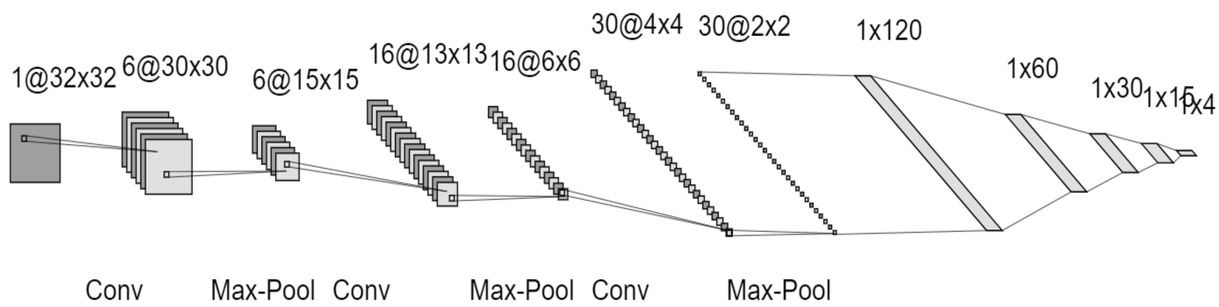


Fig: First Convolutional Neural Networks model architecture (abstract)

Model has been generated using http://alexlenail.me/NN-SVG/LeNet.html

This model architecture will be referred to as CNN-1 for simplicity in the following parts of the report

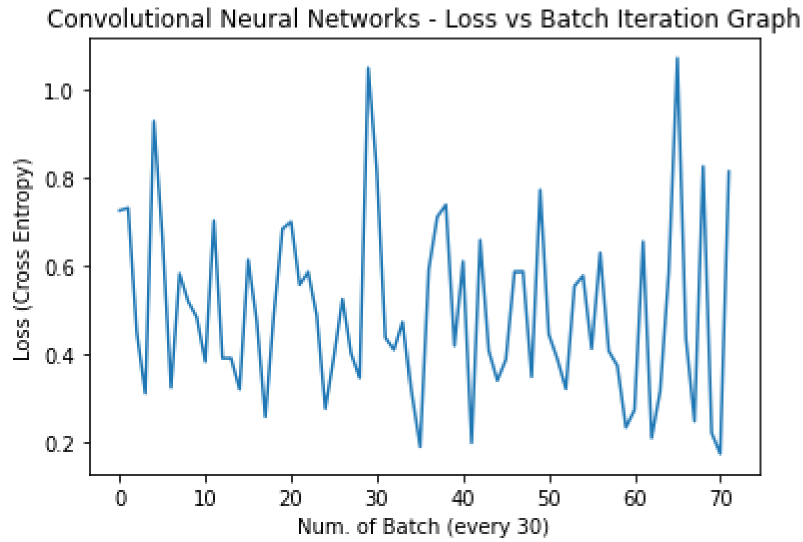
Architecture Description:

Operation	Number of Filters	Size of Each Filter	Stride	Padding	Size of Output
Input Image	-	-	-	-	32x32x1
Convolution	6	3x3	1x1	0x0	30x30x6
Dropout	-	-	-	-	30x30x6
ReLU	-	-	-	-	30x30x6
Max-Pool	1	2x2	2x2	0	15x15x6
Convolution	16	3x3	1x1	0x0	13x13x16
Dropout	-	-	-	-	13x13x16
ReLU	-	-	-	-	13x13x16
Max-Pool	1	2x2	2x2	0	6x6x16
Convolution	30	3x3	1x1	0x0	4x4x30
Dropout	-	-	-	-	4x4x30
ReLU	-	-	-	-	4x4x30
Max-Pool	1	2x2	2x2	0	2x2x30
Flatten	-	-	-	-	120
Fully Connected	-	-	-	-	60
Fully Connected	-	-	-	-	30
Fully Connected	-	-	-	-	15
Fully Connected	-	-	-	-	4

Training:

Total training sample size: 3586

The loss vs batch iteration graph for training the CNN-1 is given below:



Testing:

Total testing sample size: 896

Class	benignware	malware	hackware	gray
Samples	595	144	23	134
Predicted	708	54	0	134
Correct	579	31	0	134

Results Obtained: *F-1 Score: 0.5714022112675303*

Conclusion:

Obtained result is good but does not meet our objective. Although it is better in terms of F-1 score than the previous neural networks model.

Hyper-parameters changed for training the model:

Filter size: 5x5
Convolutional layers: 2
Fully-connected layers: 3

Model Architecture:

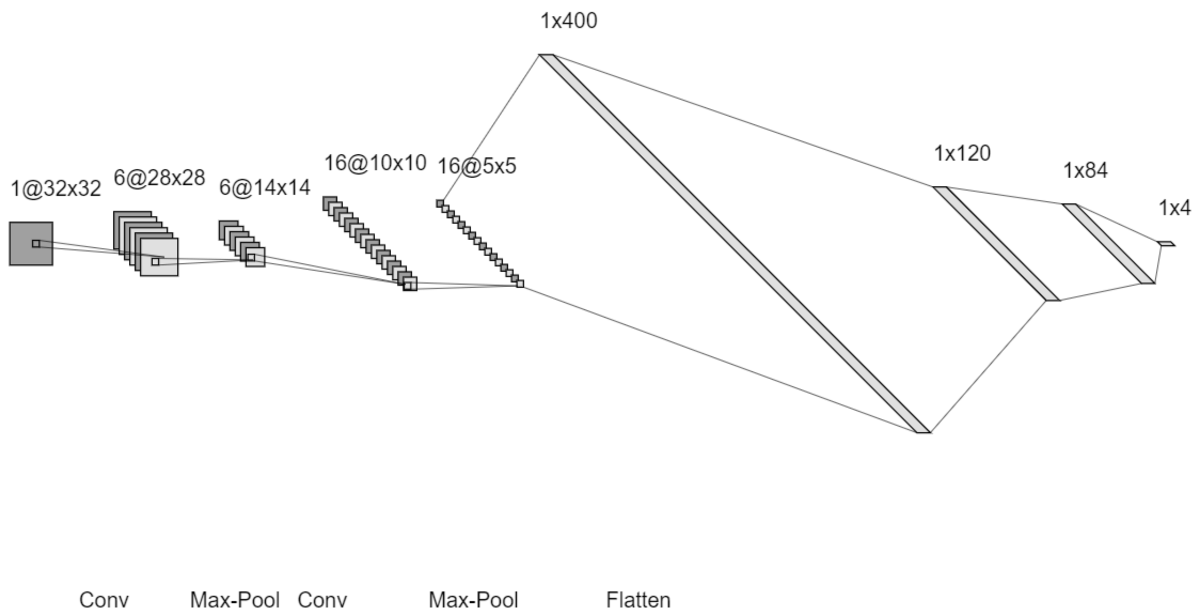


Fig: Second Convolutional Neural Networks model architecture (abstract)

Model has been generated using http://alexlenail.me/NN-SVG/LeNet.html

This model architecture is inspired by the LeNet-5 Architecture [1]. Difference with LeNet-5 is that we used dropout layers, used Max Pooling instead of Average Pooling, and for the activation function, we used ReLU instead of Sigmoid or Tanh.

This model architecture will be referred to as CNN-2 for simplicity in the following parts of the report

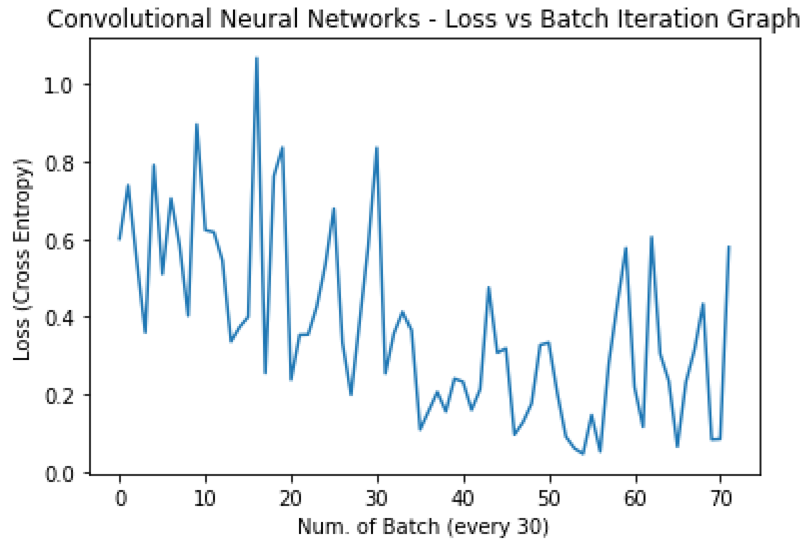
Architecture Description:

Operation	Number of Filters	Size of Each Filter	Stride	Padding	Size of Output
Input Image	-	-	-	-	32x32x1
Convolution	6	5x5	1x1	0x0	28x28x6
Dropout	-	-	-	-	28x28x6
ReLU	-	-	-	-	28x28x6
Max-Pool	1	2x2	2x2	0	14x14x6
Convolution	16	5x5	1x1	0x0	10x10x16
Dropout	-	-	-	-	10x10x16
ReLU	-	-	-	-	10x10x16
Max-Pool	1	2x2	2x2	0	5x5x16
Flatten	-	-	-	-	400
Fully Connected	-	-	-	-	120
Fully Connected	-	-	-	-	84
Fully Connected	-	-	-	-	4

Training:

Total training sample size: 3586

The loss vs batch iteration graph for training the CNN-2 is given below:



Testing:

Total testing sample size: 896

Class	benignware	malware	hackware	gray
Samples	595	144	23	134
Predicted	623	122	17	134
Correct	578	107	15	134

Results Obtained: *F-1 Score: 0.879908188340915*

Conclusion:

Obtained results have successfully fulfilled our objective (*F-1 Score to be* 0.69495 *or above*). F-1 Score also outperformed both the previous neural networks model and our CNN-1 model.

Result Analysis for both of the CNN models:

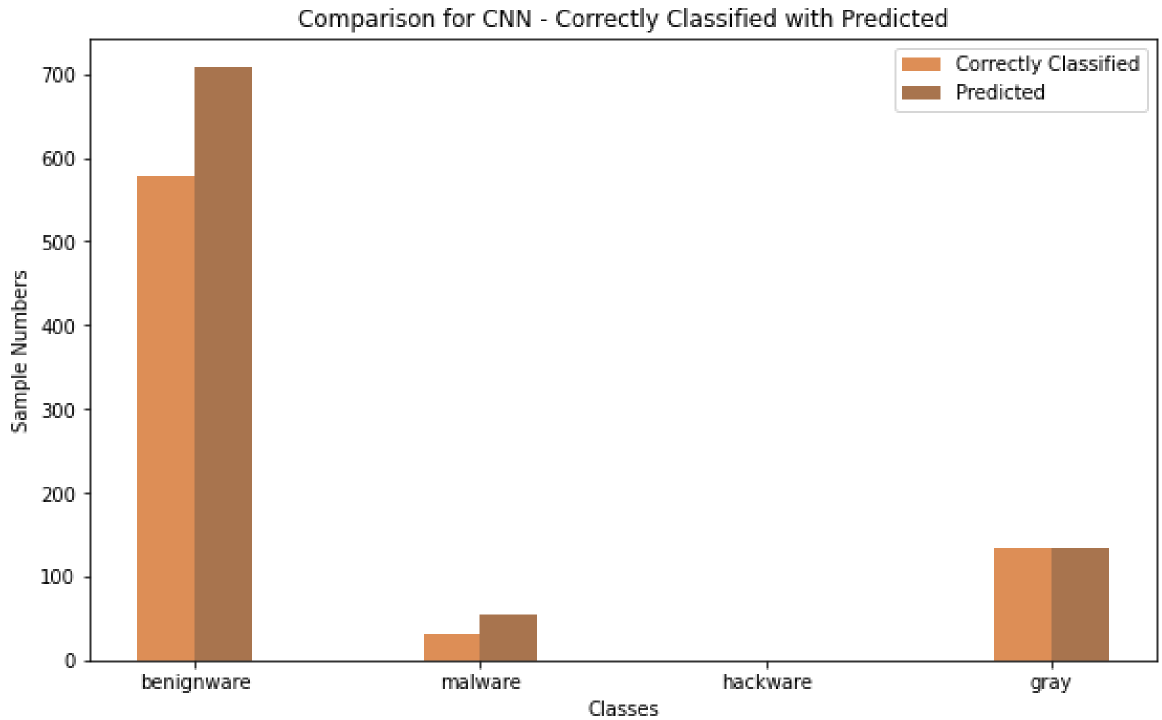


Fig: Bar diagram of classes correctly classified with their predicted times for CNN-1

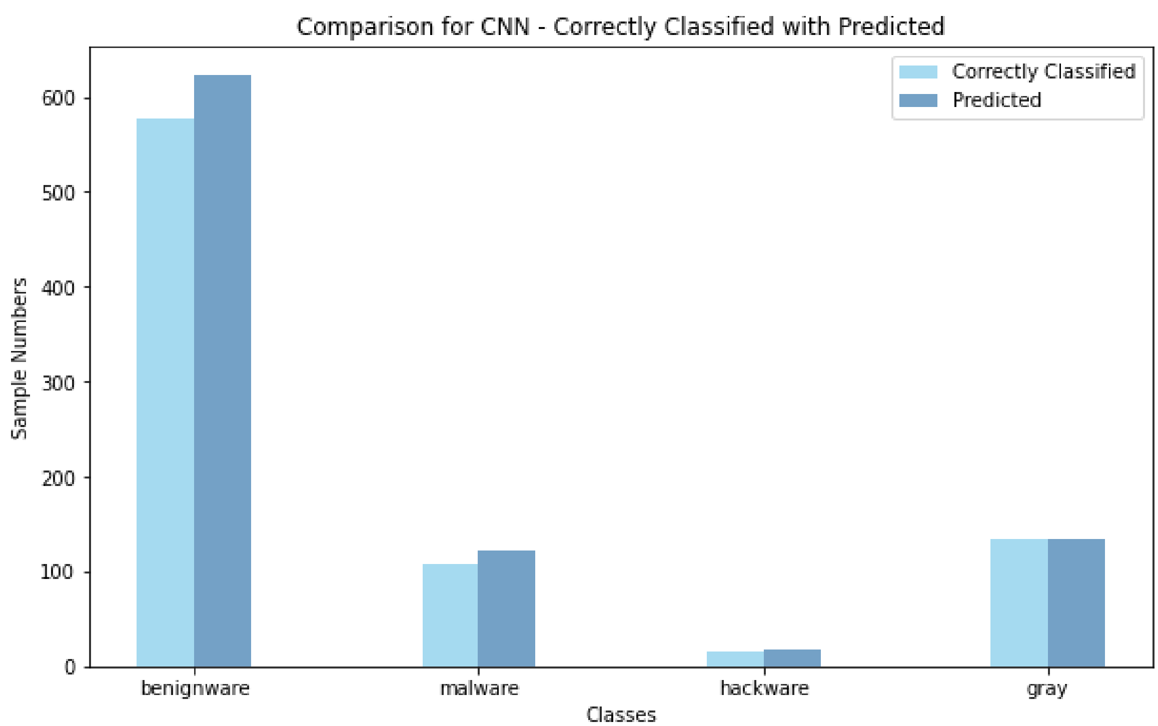
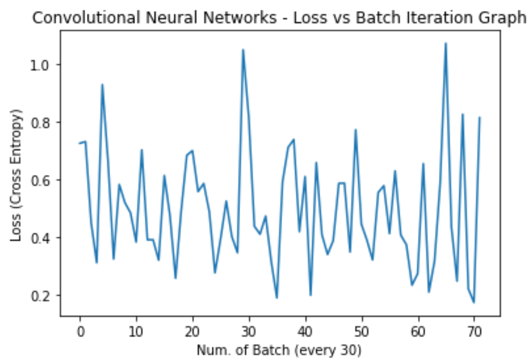
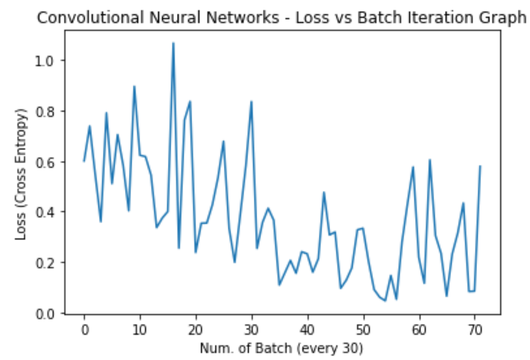


Fig: Bar diagram of classes correctly classified with their predicted times for CNN-2

If we compare the training loss of our CNN models:



CNN-1 model



CNN-2 model

The convergence curve of CNN-1 model was not quite satisfactory, but CNN-2 model's convergence was improving over each batch iteration.

If we compare the results obtained by our CNN models:

<table>
<tr><td>Class</td><td colspan="2">benignware</td><td colspan="2">malware</td><td colspan="2">hackware</td><td colspan="2">gray</td></tr>
<tr><td>Test Samples</td><td colspan="2">595</td><td colspan="2">144</td><td colspan="2">23</td><td colspan="2">134</td></tr>
<tr><td>CNN Model</td><td>1</td><td>2</td><td>1</td><td>2</td><td>1</td><td>2</td><td>1</td><td>2</td></tr>
<tr><td>Predicted</td><td>708</td><td>623</td><td>54</td><td>122</td><td>0</td><td>17</td><td>134</td><td>134</td></tr>
<tr><td>Correct</td><td>579</td><td>578</td><td>31</td><td>107</td><td>0</td><td>15</td><td>134</td><td>134</td></tr>
</table>

Both of our models performed well in terms of classifying *benignware* images. In terms of *malware* images, our CNN-2 model almost performed two times better than our CNN-1 model. For *hackware* image classification, our CNN-1 model's performance was very unsatisfactory. It could not predict any of them. And lastly, in terms of both of our CNN models, *gray* image classification had 100% accuracy, which might indicate overfitting, in this regard.

Github Link:

https://github.com/ARNoor/CSE-4238-Soft-Computing/tree/main

References:

[1] LeCun, Yann, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86.11 (1998): 2278-2324.