Predicting Malware Image Families using Convolutional Neural Network

Presented to

Dr. Fabio Di Troia

Department of Computer

Science San José State

University

In Partial Fulfillment

Of the Requirements for the

Class CS 171

By:
Eunice Oh and Rashi Raghulan

May 23, 2022

ABSTRACT

This experiment involved determining the malware family of binary malware files that have been converted to grayscale png images by training a Conventional Neural Network (CNN) using over 9000 malware images from 25 different families. We experimented with a variety of different parameters of our CNN. These experiments involved ways to preprocess the data, including shuffling and splitting, choosing the different types of layers, the number and order of these layers, as well as the number of nodes in our neural network. We also experimented with the loss functions and optimizers, and a combination of the above. Using a combination of the best parameter values we found after all of our experiments, our model produced an accuracy of 0.73 and loss of 0.73, on average. The highest accuracy our model produced was 0.78, which we believe can be improved given more time where we can conduct further experiments.

Best Parameter Values Used:

A. Random shuffling using both random.shuffle before getting first 80 images & shuffling train_test_split
B. 70-15-15 split of training-validation-testing
C. 3 Convolutional layers and 2 Dense layers, 1 Dropout layer
   a. 45 filters for the convolutional layers
   b. 55 nodes for the first dense layers
D. Categorical Cross Entropy loss function with Adam optimizer
E. Dense softmax layer with 25 nodes

Code: Google Colab Link

I. Introduction

There is a constant battle between malware detection and malware's fight to avoid antivirus detection. Therefore, malware detection is an important aspect of computer science that focuses on identifying the signatures in the malware files that help identify the file as a malware. It is important to currently study and improve malware detection as the malware viruses "mutate" or change these signatures that these detection methods identify. Therefore, a method for malware scoring that is being studied is image processing. The goal of this project is to determine which family the malware image belongs to using Conventional Neural Networks (CNN).

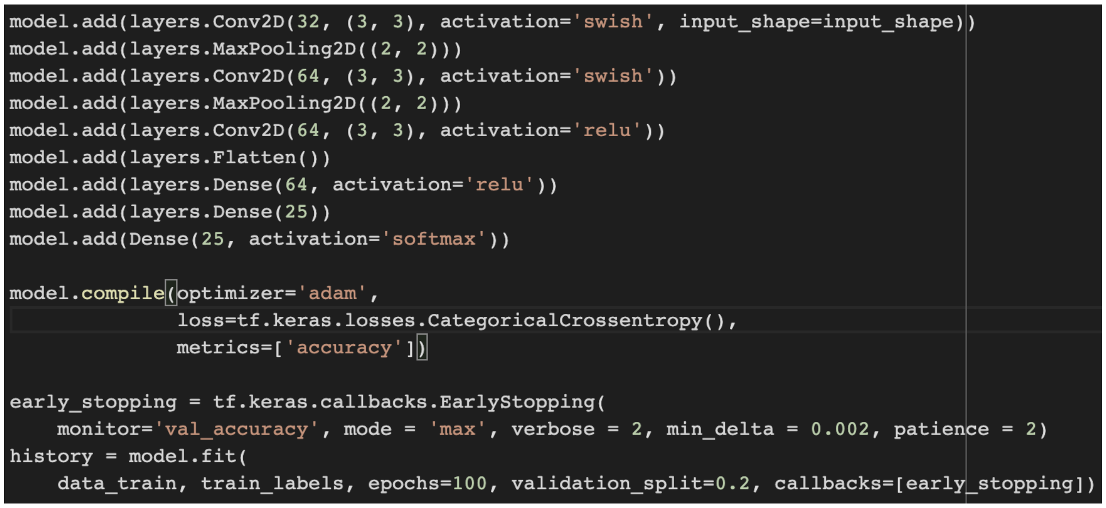
II. Description of Dataset

For this project, a dataset known as the Malimg was used which was used in the Nataraj et al., 2011 experiment. This dataset consists of 25 different families and a total of 9,339 images. The dataset consists of malware binaries for the different families that were converted into grayscale images. The malware family which had the lowest number of images was Skintrim.N with 80 images and Allaple.A was the malware family with the largest sample size of 2,949.

III. Experiments And Results

We began our experiments by first preprocessing the data. Using Google Colab, we mounted our Google Drive in order to be able to access the directory that contained the malware images. For each image in each malware family, we cropped the image in the middle, using a dimension of 64 x 208 pixels. Initially, we cropped the image from the top left corner, but we found that it led to an ambiguous data cardinality error as we were fitting our model. We suspected that cropping the images from the corner could result in some empty pixel values, which led us to crop in the center. We used 64 as our width and 208 as our height as we found these were the minimum width and height values we found for all the images, and we wanted all of our images to be the same size. We represented the images as a 2-D array of pixel values, and normalized each value to be between 0 and 1. We then balanced the data set to 80 images per malware family. To create our label list to use for training, validation, and testing, we represented each malware family as a unique number from 0 to 24. We also one hot encoded the labels to use with the categorical cross entropy loss function.

These were the specifications we used for our original base model which we used to start our experimentation: 56% Training, 14% Validation, 30% Test split, one-hot-encoded labels, 3 convolutional layers, 2 max pooling layers, categorical cross entropy as our loss function, an early stopping patience of 2, and 100 epochs. A screenshot of our base model is below:



```
model.add(layers.Conv2D(32, (3, 3), activation='swish', input_shape=input_shape))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='swish'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(25))
model.add(Dense(25, activation='softmax'))

model.compile(optimizer='adam',
              loss=tf.keras.losses.CategoricalCrossentropy(),
              metrics=['accuracy'])

early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor='val_accuracy', mode = 'max', verbose = 2, min_delta = 0.002, patience = 2)
history = model.fit(
    data_train, train_labels, epochs=100, validation_split=0.2, callbacks=[early_stopping])
```

The experiments that we are conducting are to determine the:

1. Order of Random Shuffling of Images,
2. Train / Validation / Test Splitting Ratio,
3. Number and Order of Dropout Layers,

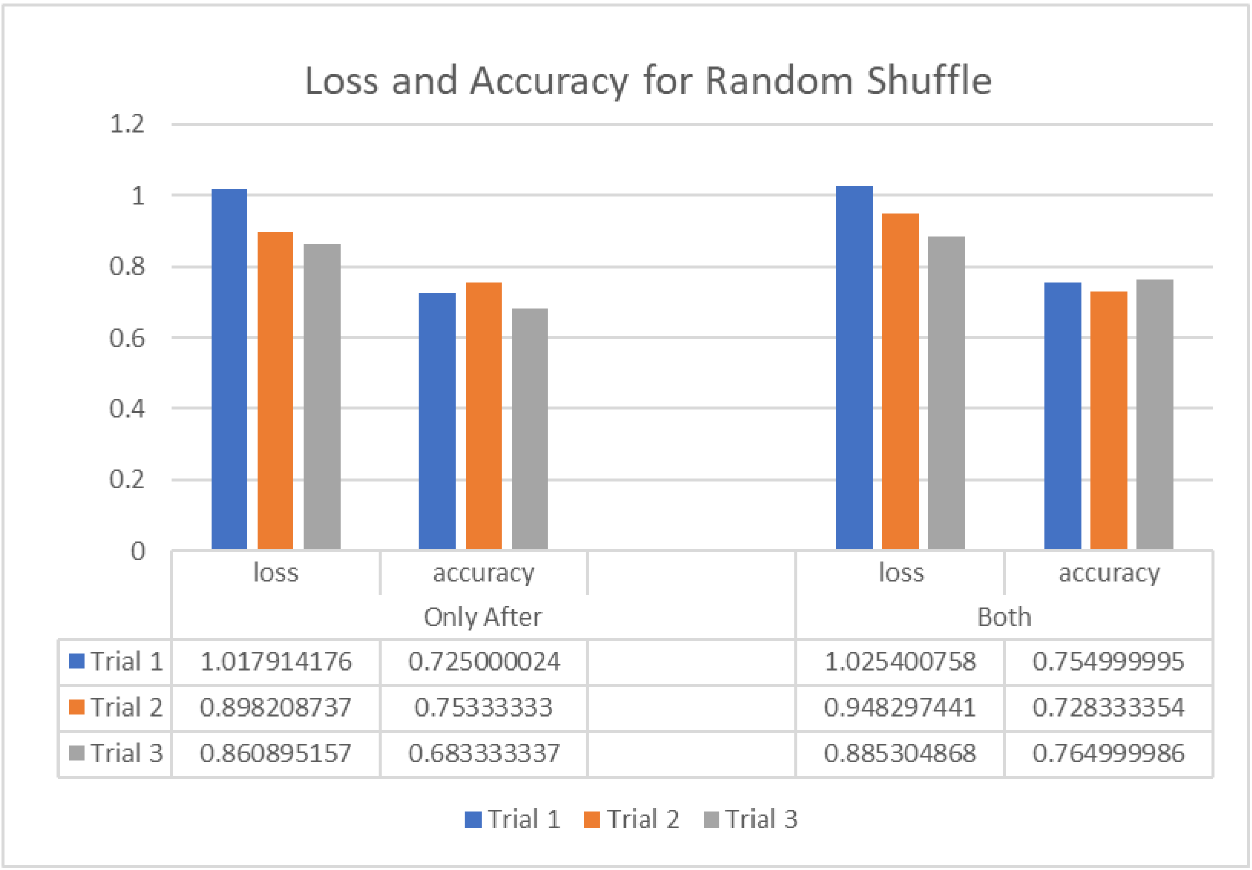
4. Combination of Dropout Layers and Train / Validate / Test split,
5. Loss Function and Optimizers,
6. Number of Convolutional Layers.,
7. Number of Filters in Convolutional Layers, and
8. Combination of Number of Dense Layers and Nodes in Dense Layers.

Our results are shown in the following images and tables.

Figure #1: Test loss and accuracy through comparison of random shuffling of images

Explanation: We experimented with shuffling the images before and after obtaining the first 80 images from the files versus random shuffling of images only after obtaining the first 80 images from the files. To randomly shuffle the images before selecting, we used the random.shuffle method to shuffle the images in each directly. Afterwards, the train_test_split method was used to create the training and testing datasets which also randomized the images.

From the 3 trials, it showed that the random shuffling using both methods produced a higher average accuracy and lower loss than just shuffling using the train_test_split method.



	loss	accuracy		loss	accuracy
	Only After			Both	
Trial 1	1.017914176	0.725000024		1.025400758	0.754999995
Trial 2	0.898208737	0.75333333		0.948297441	0.728333354
Trial 3	0.860895157	0.683333337		0.885304868	0.764999986

Figure #2: Test Accuracy vs Train / Validate / Test Split using a balanced data set of 80 images per Malimg family.

Explanation: We experimented with different training, validating, and testing split percentages, where the validating and testing data sets were split in half from the remaining data of the training split. In total, we experimented with 4 different splits, described in the order of train, validation, and test percentages: 60-20-20, 70-15-15, 80-10-10, and 90-5-5. We found that the 60 and and 80 training split had the highest accuracy of 74.5%.

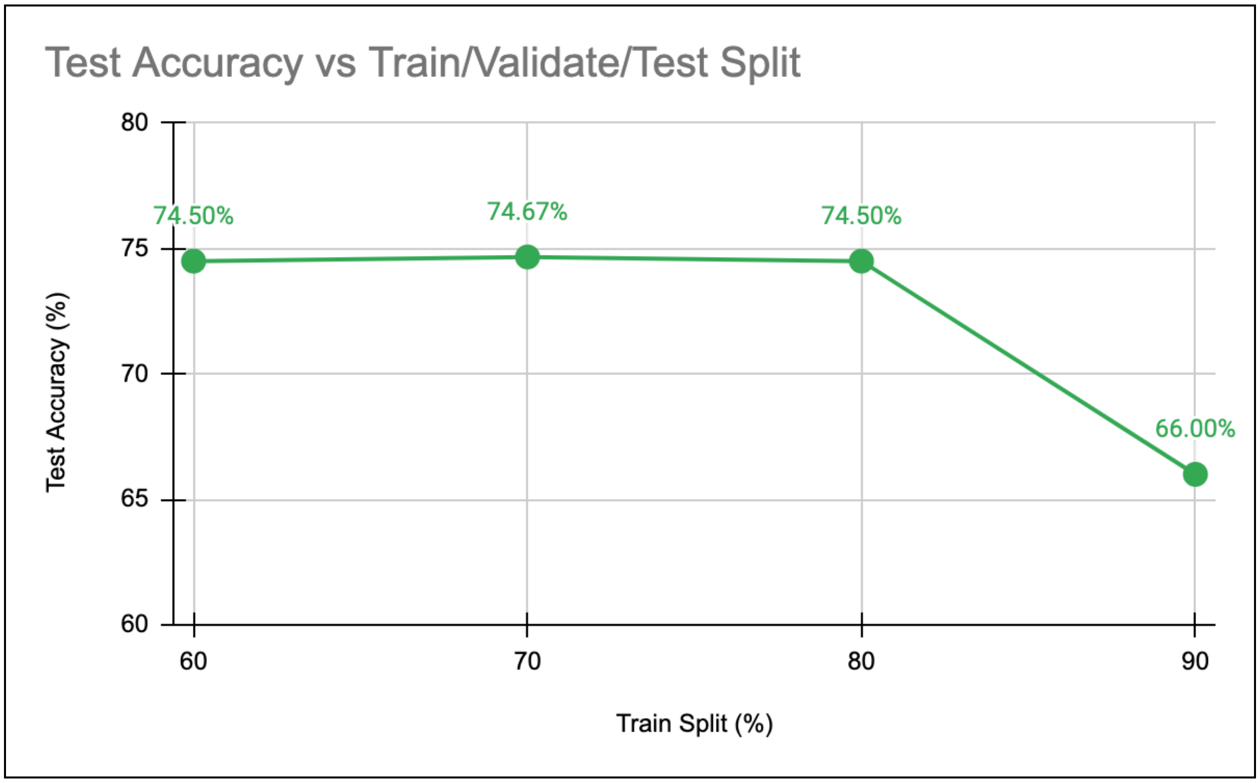


Figure #3: Test Accuracy vs Dropout Layers using a balanced data set of 80 images per Malimg family.

Explanation: We experimented with the number and order of dropout layers to add to our convolutional neural network. The CNN used to test with the dropout layers involved 4 convolutional and 4 pooling layers, compared to our base model of 3 convolutional and 2 pooling layers. We added more layers in order to experiment with a greater number and order of dropout layers. The number of flatten and dense layers remained the same. The order and number of dropout layers was represented by a binary vector. A ‘1’ means we added a dropout layer, and a ‘0’ means we did not add a dropout layer in the given position. The order of the dropout layers was after the first 2 convolution & pooling layers, after the first 4 convolutional & pooling layers, and after the first dense layer. For example, a [1,0,1] means we added a dropout layer after the first 2 convolution & pooling layers and after the first dense layer, but not after the first 4 convolutional & pooling layers. The dropout percentage we used for the first 2 layers was 20%, and the dropout percentage for the 3rd layer was 50%. We decided to use a higher dropout percentage for the dense layers as we learned that was the suggested dropout percentage. We used a smaller percentage as we learned 50% may be too extreme, and a smaller value may work better. A future experiment may be to experiment with the different dropout percentages as well. We learned that the dropout order of [0,1,1] produced the highest test accuracy of 74.17%.

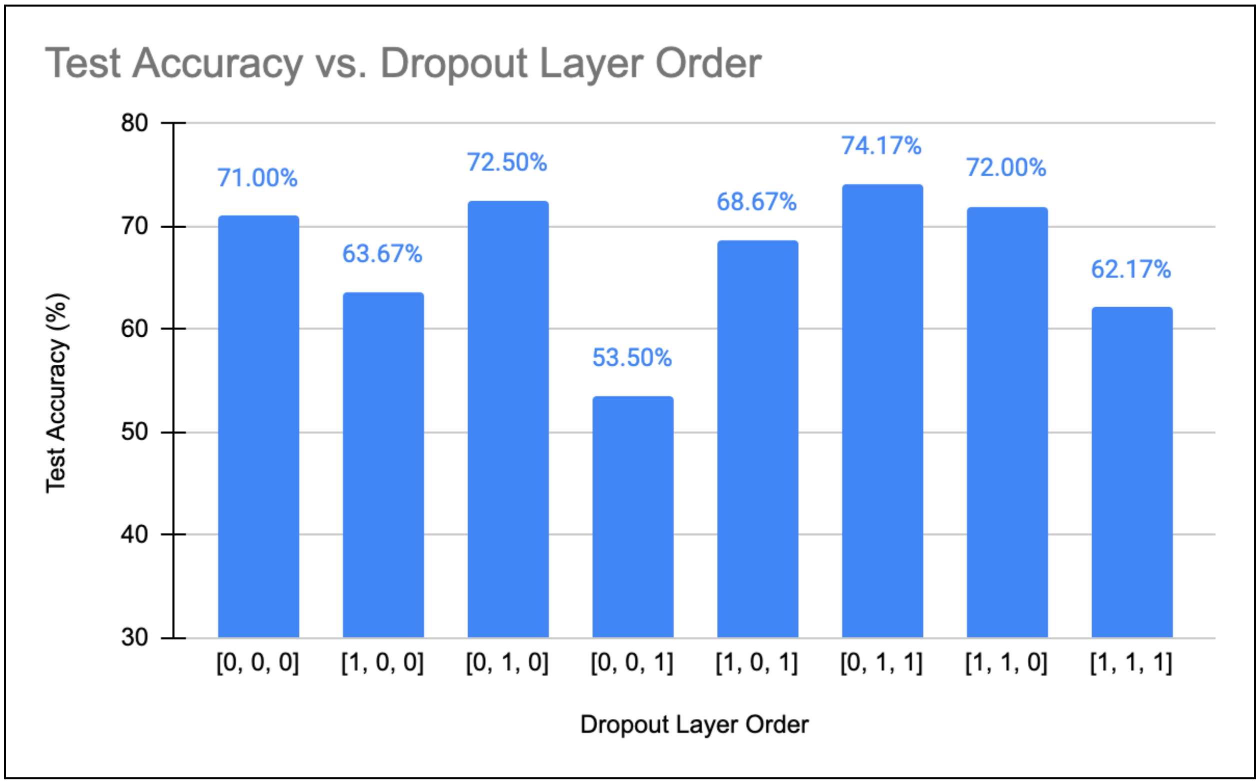


Figure #4: Test Accuracy vs a combination of Dropout Layers and Train / Validate / Test split using a balanced data set of 80 images per Malimg family.

Explanation: We experimented with the order of dropout layers with 4 different split variations. For each split that we previously tested in Figure #2, we applied the same dropout layer order and combination that we previously tested in Figure #3. We found that a 70-15-15 train / validate / test split with a dropout layer order of [0,1,0] had the best test accuracy of 78%.

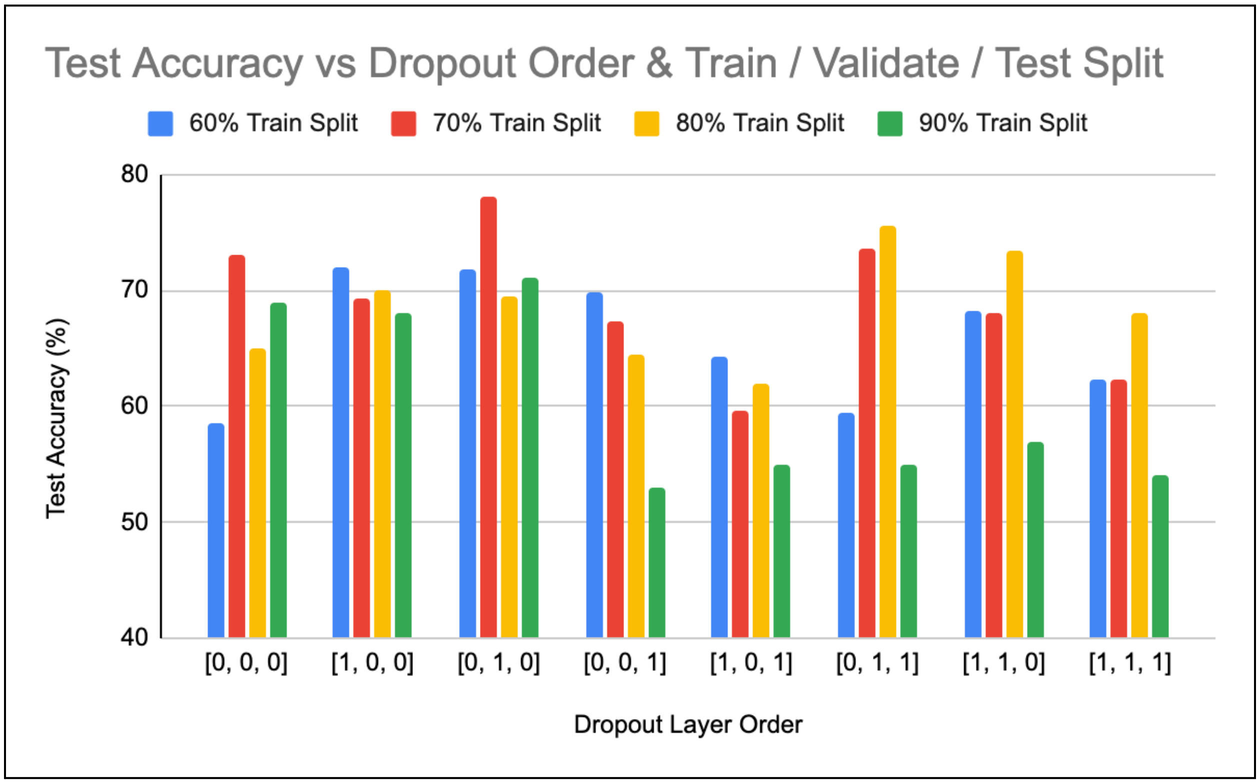


Figure #5: Test Accuracy vs Optimizer vs Loss Function using a balanced data set of 80 images per Malimg family.

Explanation: We also experimented with 2 different loss functions, categorical cross entropy and sparse categorical cross entropy, and 3 different optimizers, adam, adagrad, and RMSprop (using the base model). For each loss function, we applied the 3 optimizers. When using the categorical cross entropy, we one hot encoded our labels, but skipped doing so for the sparse categorical cross entropy, as sparse does not require the labels to be one hot encoded. We found that the combination of using the categorical cross entropy loss function and the adam optimizer resulted in the highest test accuracy of 75.83%.

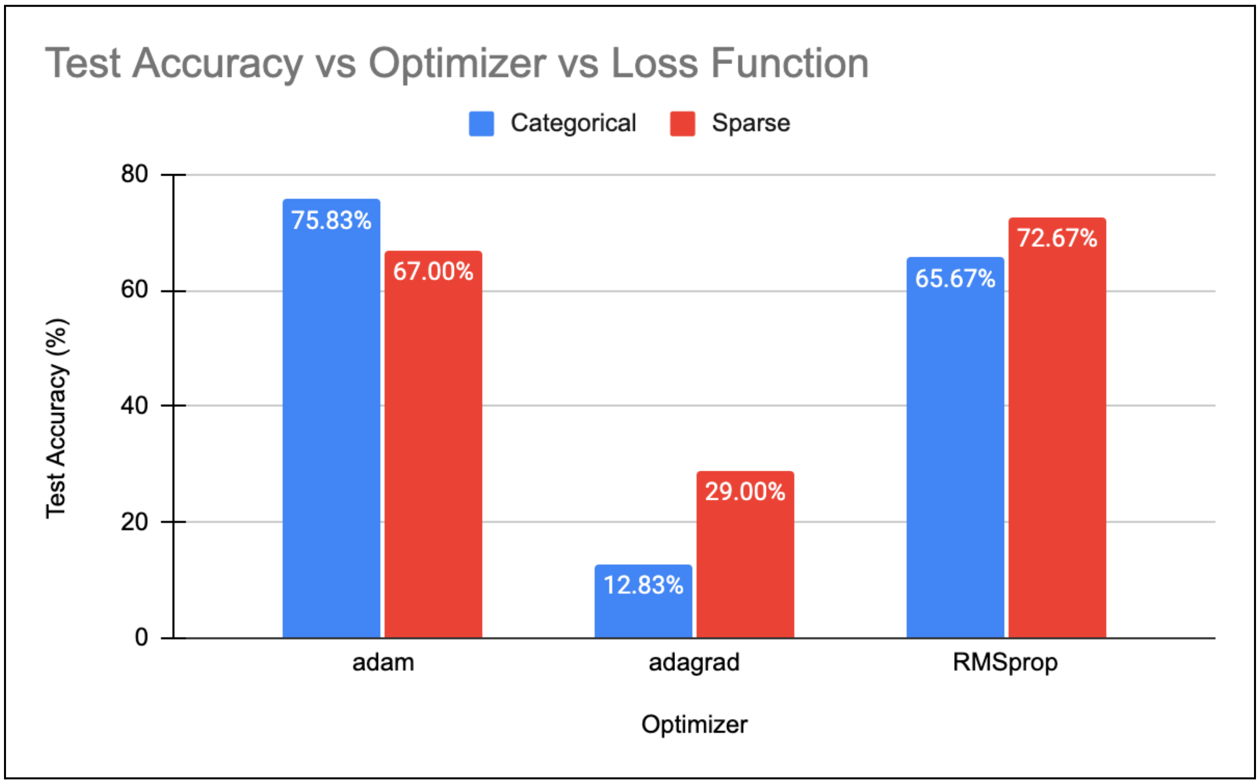
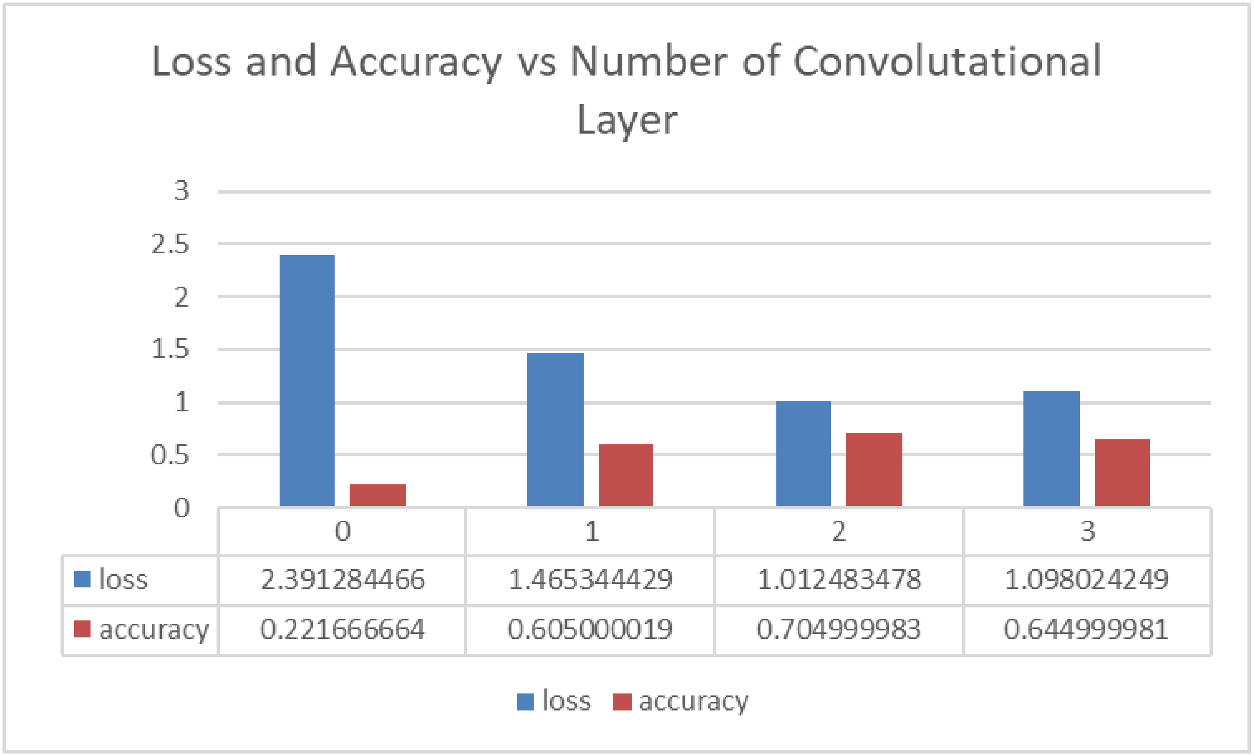


Figure #6: Test Loss and Accuracy Across Several Convolutional Layers

Explanation: Next, we tested the number of convolutional layers required. The bar graph below shows that two convolutional layers (after one input convolutional layer) or three total convolutional layers are required to produce the highest accuracy of 0.705 and a loss of 1.0125.

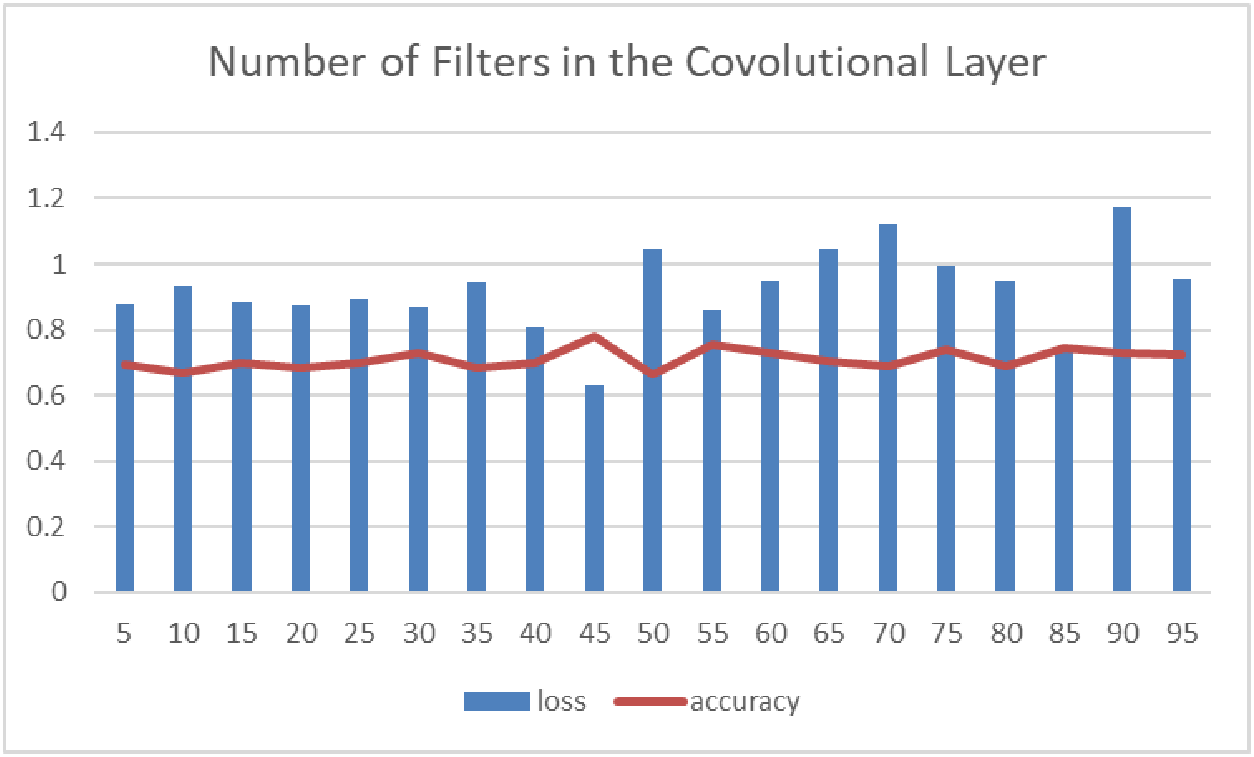


	0	1	2	3
🟦 loss	2.391284466	1.465344429	1.012483478	1.098024249
🟥 accuracy	0.221666664	0.605000019	0.704999983	0.644999981

Figure #7: a) Number of Filters in the Convolutional Neural 2D Layer over Increments of 5 b) Number of Filters in the Convolutional Neural 2D Layer over 10 different layers

Explanation: We used a range of 5-95 different filters to determine the number of filters required in each of the convolutional neural layers. The highest accuracy or the accuracy of 0.78 was obtained with 45 filters with the loss of 0.6316.

To determine the exact number of filters required, a range of 10 filters was used. The highest accuracy was still obtained with 45 filters which produced an accuracy of 0.75 and a loss of 0.7842.


Number of Filters in the Covolutional Layer
1.4
1.2
1
0.8
0.6
0.4
0.2
0
5
10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
90
95
loss
accuracy

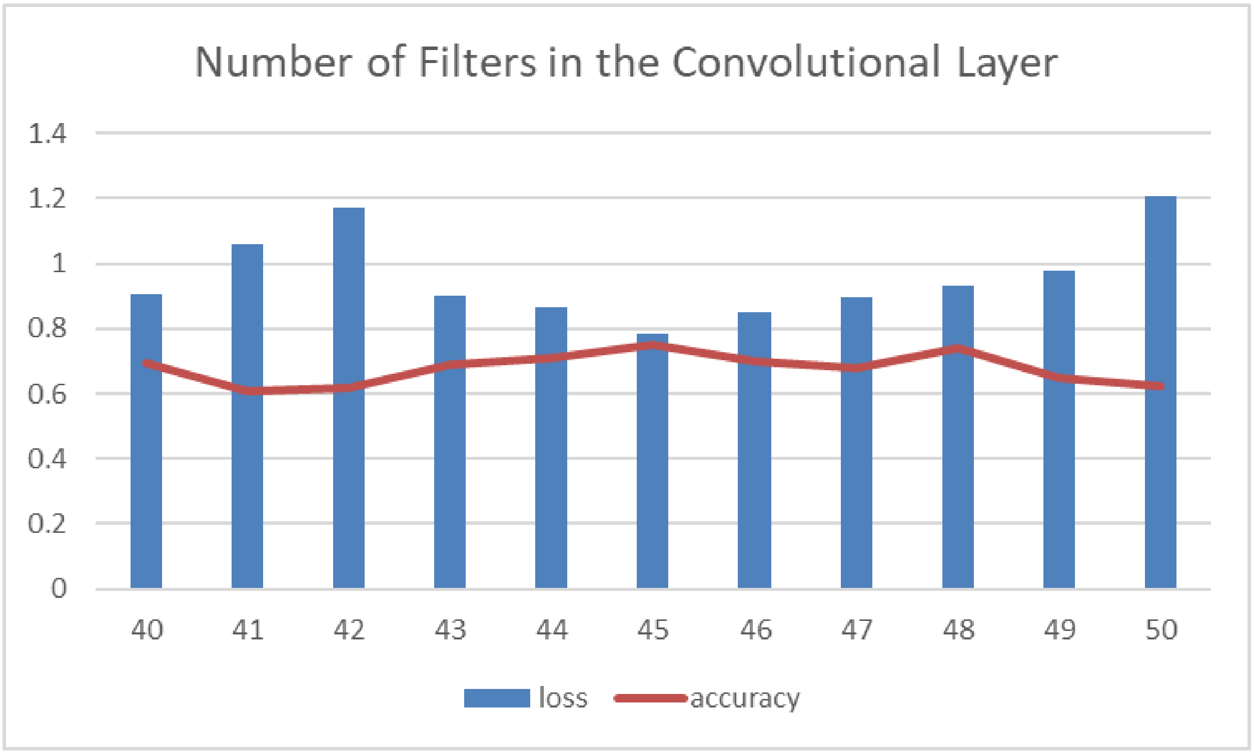
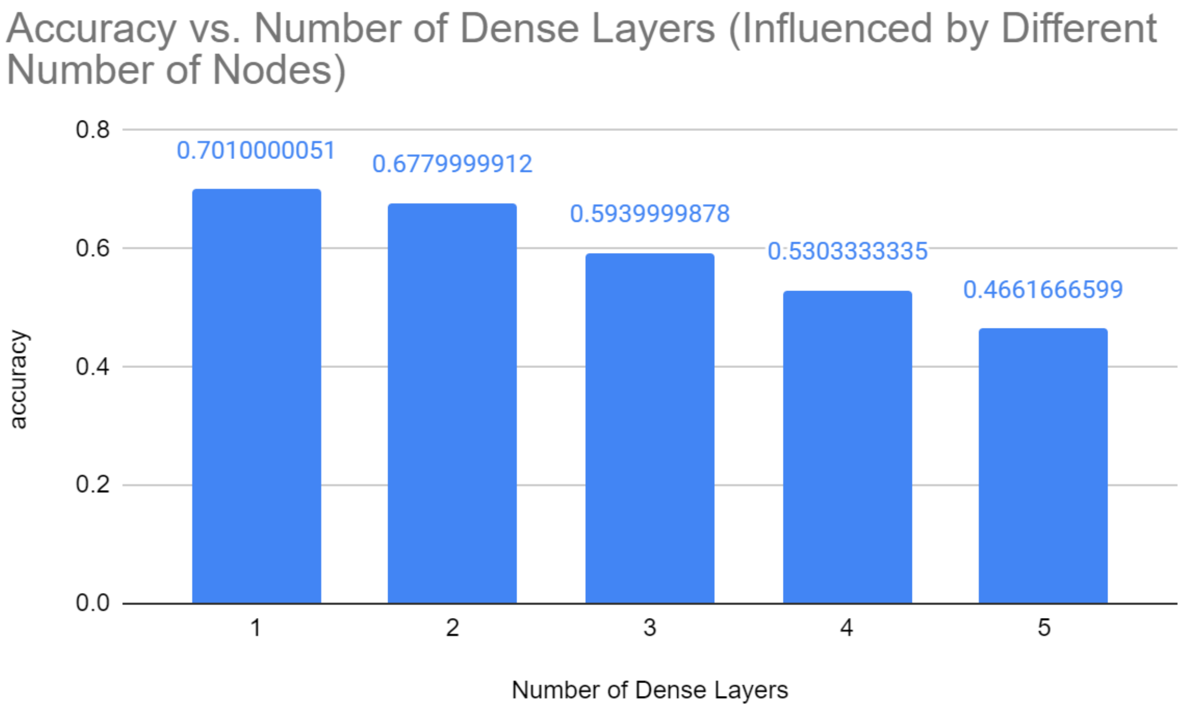
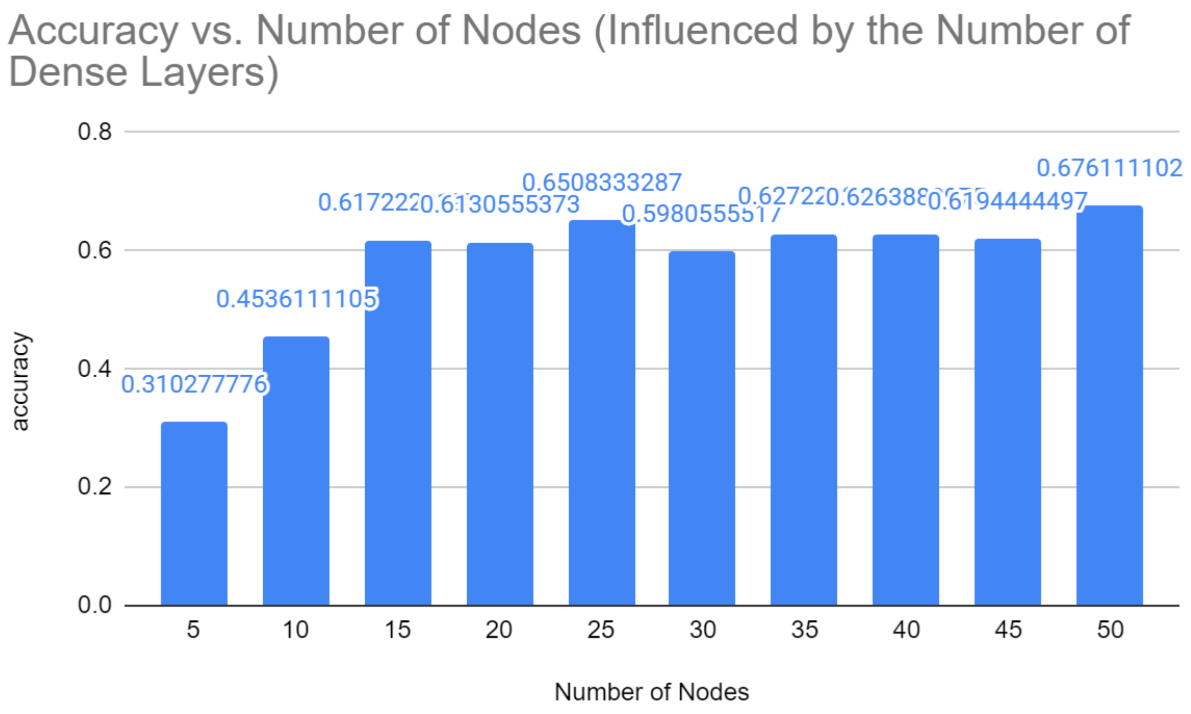

Number of Filters in the Convolutional Layer
1.4
1.2
1
0.8
0.6
0.4
0.2
0
40
41
42
43
44
45
46
47
48
49
50
loss
accuracy

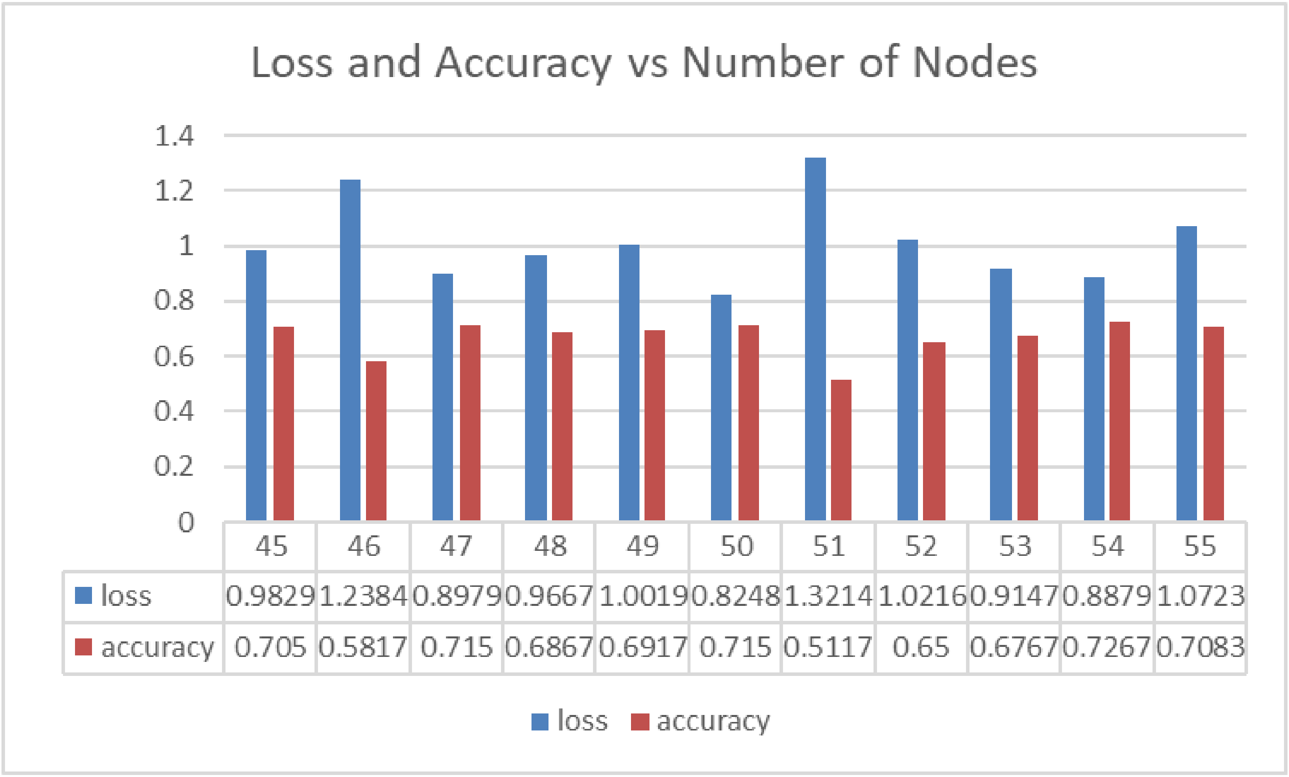
Figure #8: Test accuracy across several dense layer and several nodes

Explanation: We experimented with the different number of dense layers and the number of nodes in the dense layers. There were two for loops so both variables were influenced by each other and bar graphs below show the average accuracy. For example, the leftmost bar on the Accuracy vs. Number of Dense Layers graph shows the average accuracy for all the different nodes: 5 to 50 which incremented with a range of 5 when there was only one dense layer in the model.

As the results show the best averages were obtained when there was only one dense layer with 50 average nodes followed by the last softmax dense layer of 25 nodes or two dense layers, in total. All of the information is shown below in the appendix.







	45	46	47	48	49	50	51	52	53	54	55
■ loss	0.9829	1.2384	0.8979	0.9667	1.0019	0.8248	1.3214	1.0216	0.9147	0.8879	1.0723
■ accuracy	0.705	0.5817	0.715	0.6867	0.6917	0.715	0.5117	0.65	0.6767	0.7267	0.7083

IV. Conclusion and Future Works

After all of our experiments, we determined that random shuffling using both random.shuffle before getting the first 80 images and shuffling using training and testing produced the best accuracy. Next, we determined that we should conduct a 70-15-15 split of training-validation-testing to produce the best results with a dropout layer order of [0,1,0] to avoid overfitting. We also determined that using the Categorical Cross Entropy loss function and Adam optimizer produced the lowest loss (and probably reached the global minimum). Next, we determined that the number of convolutional layers and dense layers must be a total of 3 and 2, respectively for our experiment with 45 filters for the convolutional layers and 55 nodes for the first dense layers followed by a dense softmax layer with 25 nodes.

There are several experiments that we can still conduct. While we had determined the number of convolutional layers and the number of filters for each layer, we made an assumption that the number of filters in every convolutional layer is the same but we can conduct experiments and determine the filters per layer. Moreover, we can also play around with the other parameters in the convolutional layers such as the filter size and stride length which was kept constant in our experiments. Similarly, we can also play around with the size of the pooling layer. Moreover, we can also test the different activation functions in all of the layers instead of keeping it just 'relu'. Moreover, we found the number of convolutional layers and dense layers by keeping one constant and changing the other; however, our accuracy might increase by conducting an experiment which combines the convolutional layer experiment and the dense layer experiment. Lastly, similar to the paper, we can also try to use GIST descriptors to improve our accuracy.

V. References

1. Yajamanam, Sravani & Selvin, Vikash & Di Troia, Fabio & Stamp, Mark. (2018). Deep Learning versus Gist Descriptors for Image-based Malware Classification. 553-561. 10.5220/0006685805530561.

2. Nataraj, L., Karthikeyan, S., Jacob, G., and Manjunath, B. S. (2011). Malware images: Visualization and automatic classification. In Proceedings of the 8th International Symposium on Visualization for Cyber Security, VizSec '11, pages 4:1–4:7, New York, NY, USA. ACM.

3. *Dropout rate guidance for hidden layers in a convolution neural network*. Stack Overflow. Retrieved May 23, 2022, from https://stackoverflow.com/questions/47892505/dropout-rate-guidance-for-hidden-layers-in-a-convolution-neural-network

VI. APPENDIX

Table for Figure #8: Test accuracy across several dense layer and several nodes

Number of Nodes	5	10	15	20	25	30	35	40	45	50
Number of Dense Layers	0	0	0	0	0	0	0	0	0	0
loss	0.829891	0.807733	1.207357	0.856665	0.738785	0.909243	0.845735	0.823035	1.391294	0.779023
accuracy	0.741667	0.72	0.578333	0.713333	0.718333	0.721667	0.728333	0.733333	0.596667	0.758333
Number of Nodes	5	10	15	20	25	30	35	40	45	50
Number of Dense Layers	1	1	1	1	1	1	1	1	1	1
loss	0.866822	0.929866	0.792008	0.844555	1.000985	1.109563	1.01793	1.242906	0.911837	1.051633
accuracy	0.715	0.706667	0.731667	0.715	0.69	0.585	0.705	0.548333	0.708333	0.675
Number of Nodes	5	10	15	20	25	30	35	40	45	50
Number of Dense Layers	2	2	2	2	2	2	2	2	2	2
loss	2.891693	1.202629	1.040079	1.256641	1.086934	1.07343	0.997337	0.926907	1.254533	0.998327
accuracy	0.125	0.583333	0.701667	0.586667	0.671667	0.701667	0.648333	0.696667	0.58	0.645

Number of Nodes	5	10	15	20	25	30	35	40	45	50
Number of Dense Layers	3	3	3	3	3	3	3	3	3	3
loss	2.690594	1.892326	1.635725	1.25909	1.152544	1.295092	1.143666	1.218941	1.322462	0.938095
accuracy	0.116667	0.375	0.49	0.566667	0.648333	0.585	0.608333	0.656667	0.555	0.701667
Number of Nodes	5	10	15	20	25	30	35	40	45	50
Number of Dense Layers	4	4	4	4	4	4	4	4	4	4
loss	3.213741	2.913764	1.064014	1.14012	1.270436	1.30708	1.480636	1.588069	1.098649	1.266325
accuracy	0.013333	0.133333	0.65	0.578333	0.575	0.505	0.503333	0.451667	0.658333	0.593333
Number of Nodes	5	10	15	20	25	30	35	40	45	50
Number of Dense Layers	5	5	5	5	5	5	5	5	5	5
loss	2.77413	2.494737	1.405731	1.35376	1.305518	1.379517	1.258121	1.083195	1.168029	0.922477
accuracy	0.15	0.203333	0.551667	0.518333	0.601667	0.49	0.57	0.671667	0.618333	0.683333