A Comparison of Regularization Techniques in DNN

Sampad Kumar Kar
S. Aslah Ahmad Faizi

January 10, 2022

1 Introduction

Artificial Neural Networks (ANN) have garnered significant attention from researchers and engineers because of its ability to solve many complex problems with reasonable ability. If enough data are provided during the training process, ANNs are capable of achieving good performance results. However, if training data are not enough, the predefined neural network model suffers from overfitting and underfitting problems. To solve these problems, several regularization techniques have been devised and widely applied to ANNs. However, it is difficult to choose the most suitable scheme for ANNs because of the lack of information regarding the performance of each of these schemes while competing against each other.

2 Problem

This paper describes a comparative research on several popular regularization techniques by evaluating the training and validation errors in a DNN model, using a real world weather dataset.
At the end the paper also tests this on their own test set, to validate the conclusions based on training and validation errors, to come up with the best regularization paradigm.

3 Experiment

The goal is to compare the performance of several powerful regularization methods, which includes autoencoder, data augmentation, batch normalization, and regularization.
First, the dataset is tested on a DNN without using any regularization methods. This is trained and validated and the errors are calculated and noted. Next, the errors are calculated using the same model, without changing the settings, after applying regularization schemes.
Then each regularization scheme is analyzed by comparing the experiment results. The entire experiment consists of 9 steps. Each step is described in the Model Details section. The framework can be seen in Figure 1.

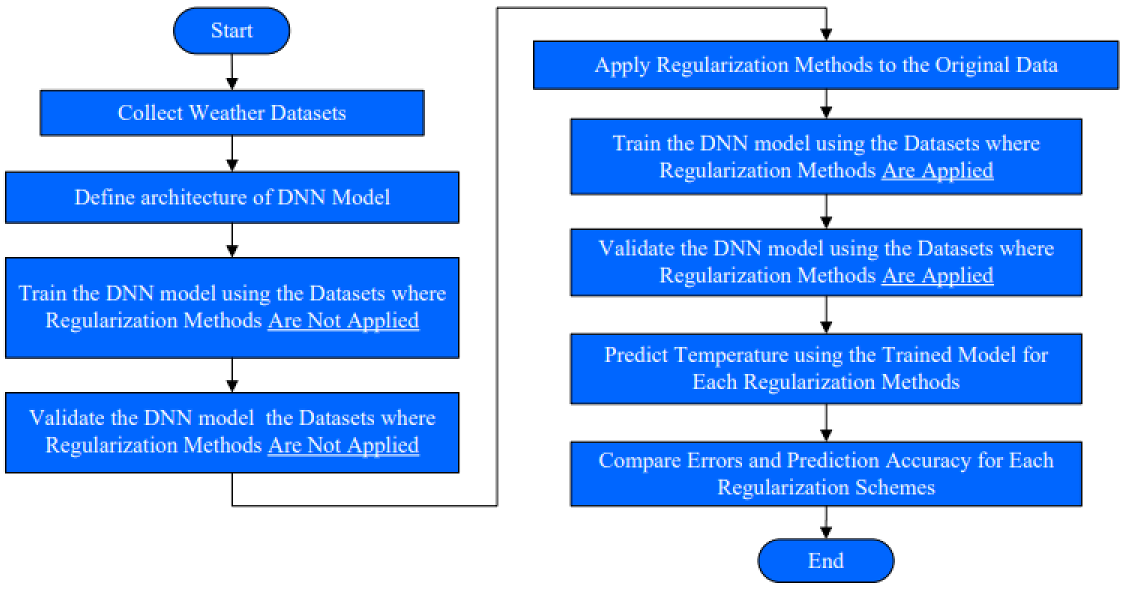

Start
Collect Weather Datasets
Define architecture of DNN Model
Train the DNN model using the Datasets where Regularization Methods Are Not Applied
Validate the DNN model the Datasets where Regularization Methods Are Not Applied
Apply Regularization Methods to the Original Data
Train the DNN model using the Datasets where Regularization Methods Are Applied
Validate the DNN model using the Datasets where Regularization Methods Are Applied
Predict Temperature using the Trained Model for Each Regularization Methods
Compare Errors and Prediction Accuracy for Each Regularization Schemes
End

Figure 1: The methodology used

4 Setup

The hardware and software used for each of these was kept the same.
CPU - Intel Core i7-4790 K (no GPU processing used for computation)
RAM - 8 GB
OS - Windows 10
Programming Language - Python
Libraries - TensorFlow and Sci-Kit Learn
Visualization - Matplotlib

5 Model Details

5.1 Datasets

The weather dataset that were used to train and validate the DNN were collected from the Korean government website.
No information regarding the no. of samples used were given in the paper, but each sample has 35 features, which consists of 7 features (average temperature, maximum temperature, minimum temperature, average wind speed, average humidity, cloudiness, and daylight hours) for 5 consecutive days (which gives 5*7 = 35 features in total).
The target label was the average temperature of the next day.

5.2 Architecture

The basic model included 1 input layer, 2 hidden layers and 1 output layer. The input layer contained 35 neurons, the hidden layers contained 50 neurons each and the output layer had 1 neuron. The model followed a 35-50-50-1 topology.

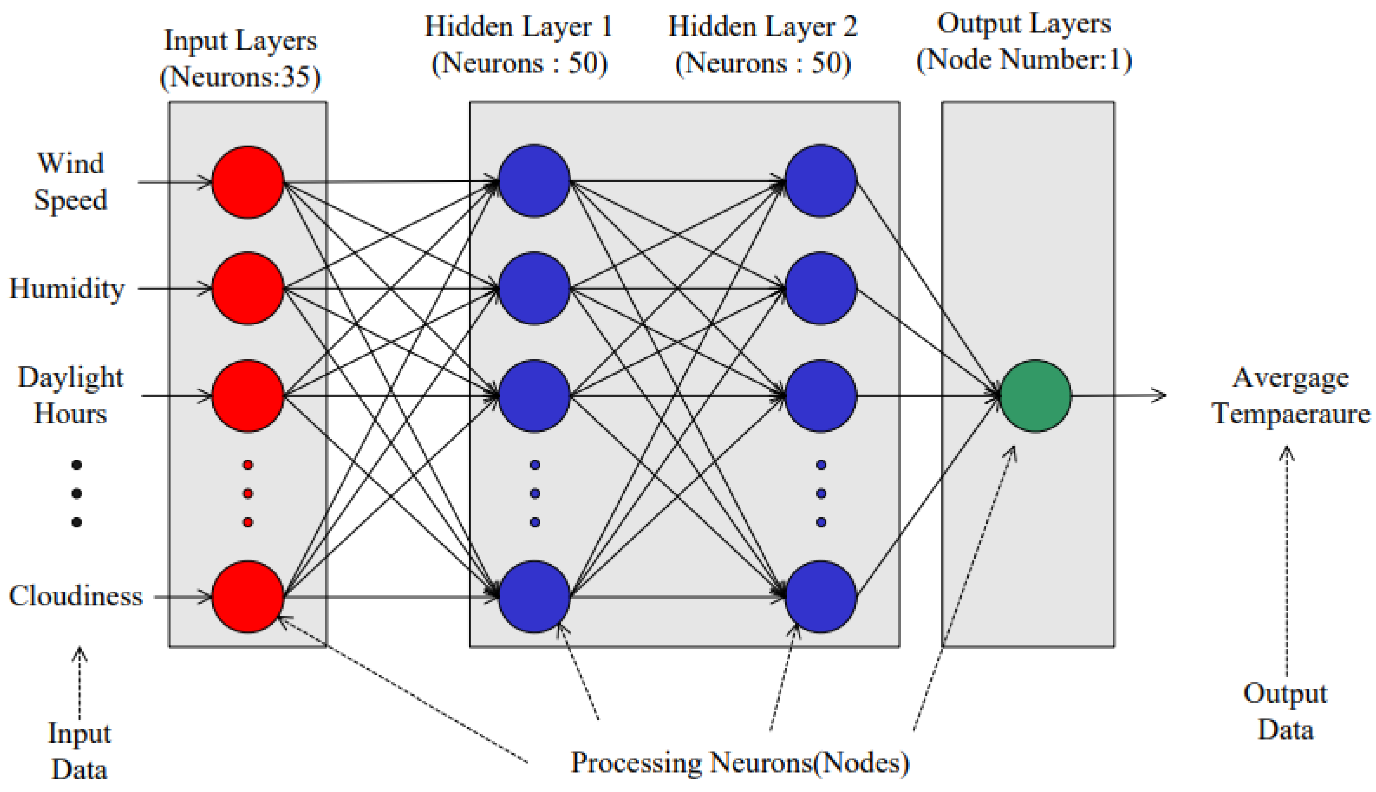

Input Layers (Neurons:35)
Hidden Layer 1 (Neurons : 50)
Hidden Layer 2 (Neurons : 50)
Output Layers (Node Number:1)
Wind Speed
Humidity
Daylight Hours
Cloudiness
Input Data
Avergage Tempaeraure
Output Data
Processing Neurons(Nodes)

Figure 2: The NN model applied

6 Hyperparameters

During this, the learning rate was set to 0.0001, which was slightly slow but reliable. The 'proximal Adagrad optimizer' algorithm was used to optimize the model. (Here is its documentation)
The batch size was 100 and epoch size was 100,000.
The activation function used was 'rectified linear unit' (ReLU).
For computing the errors during training and validation, the RMSE (root mean squared errors) were used via the following equation:

$$RMSE(NN_{Model}) = \sqrt{\left(\frac{1}{m}\sum_{i=1}^{m}\left(\frac{Answer(X_i) - Predict(X_i)}{Answer(X_i)}\right)^2\right)}$$

where, $Answer(X_i)$ is the actual label for the i^{th} example and $Predict(X_i)$ is the predicted output by the DNN.

Table 1. Parameter settings applied

Parameters for Each Model	Typical DNN	Autoencoder	Data Augmentation	L1 Regularization	Batch Normalization
Number of input neurons	35	35	35	35	35
Number of hidden layers	2	2	2	2	2
Number of neurons in hidden layers	50	50	50	50	50
Number of output neurons	1	1	1	1	1
Learning rate	0.0001	0.0001	0.0001	0.0001	0.0001
Activation function	ReLU	ReLU	ReLU	ReLU	ReLU
Optimizer	Proximal Adagrad	Proximal Adagrad	Proximal Adagrad	Proximal Adagrad	Proximal Adagrad

These are repeated over the regularized DNN as well, keeping the parameter and data the same, so that the comparison can be done without any biases.

7 Comparing Regularization Techniques

Since overfitting and underfitting problems are much more visible in pictures, our final results are visualized as graphs. The axes of the graphs consist of RMSE error values and epoch numbers. Even after training the network model, very low RMSE values seemed to be very good accuracy. However, in some cases, they caused issues such as an overfitting problem.
Typical Inference after observing the RMSE Training and Validation Loss.

Symptoms	Diagnosis
Training Loss - low Validation Loss - low	Just the right fit
Training Loss - high Validation Loss - high	Underfitting
Training Loss - low Validation Loss - high	Overfitting

7.1 Without Regularization

The above defined DNN is trained without applying any regularization methods. During training, the RMSEs were captured and the datas were saved into a separate file. The model was established with the settings as shown in Table 1.
After training 100,000 epochs, RMSEs for training and validation data were plotted, as shown below in Figure 3 and Figure 4 respectively. Figure 3 shows that by increasing the epochs, the error for the training data was prone to changing rapidly. However, the overall training error also decreased.
By comparing the results, we concluded that the errors for validation and training data decreased in the same range. The closeness between the validation and training data errors meant that good generalization was achieved.

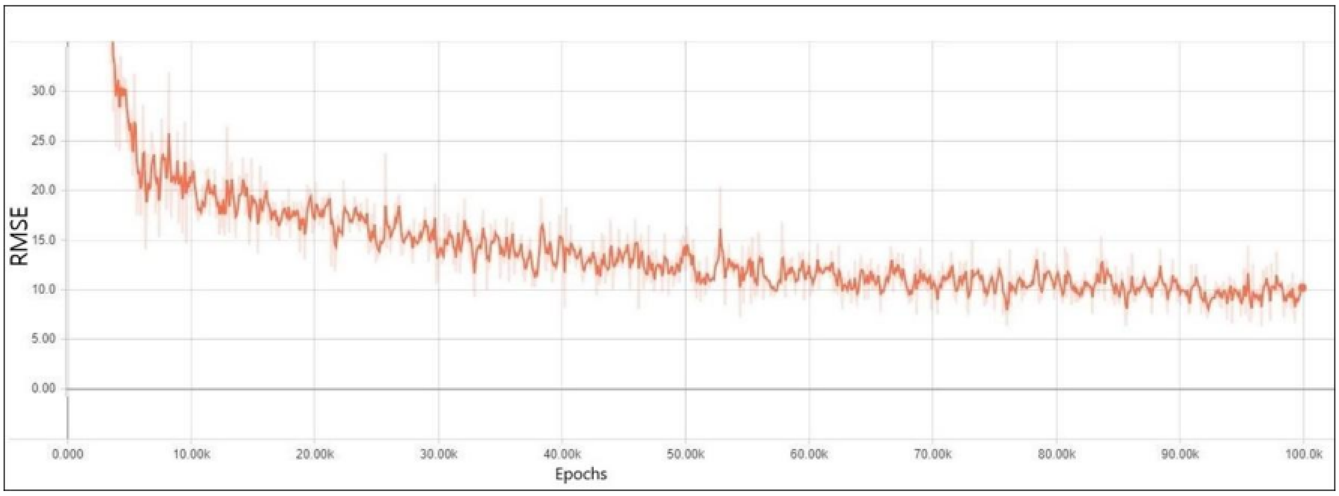


Figure 3: Training error without regularization methods

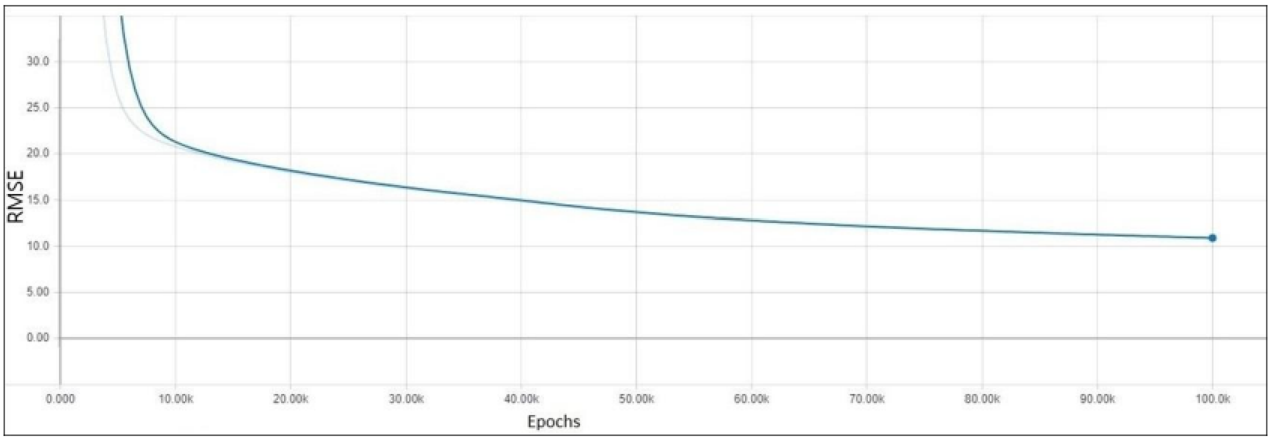


Figure 4: Validation error without regularization methods

7.2 Autoencoders

For the next model, we use the autoencoder scheme. Details regarding the implementation of autoencoder are not provided in the paper, so we are presenting the inference based on the limited information provided in the paper.

Autoencoder refers to a Neural Network with the same number of neurons in the input as well as the output layers. This scheme uses unsupervised learning, because we are not using any labels while training the autoencoder. The scheme compresses the data received from the input neurons into short code, and then decompresses this code into output neurons that are very close to the input data.

One of the goals of this scheme is to remove the noise from the input data. By removal of noise from the dataset, we mean to remove the unwanted data items/features, that causes the algorithm to miss out patterns in the data.

The architecture of autoencoders is similar to MLP (Multi Layer Perceptrons), it has at least 1 input, hidden and output layer. This type of neural network consists of two parts, an encoder and a decoder. An encoder is a network component that compresses the input. A decoder is used to reconstruct the

encoded input.
For example, a simple autoencoder with a single encoder and decoder layer has its encoder take in an input $x \in X$ and compress it to $z \in Z$, via the equation:

$$z = \sigma\left(\langle W, x\rangle + b\right)$$

where, σ is the activation function (ReLU, in our case) and W is the weight of the nodes in the encoder layer and b is the bias vector.
In the reconstruction process, the same template is repeated:

$$x' = \sigma'\left(\langle W', z\rangle + b'\right)$$

where, σ', W', b' are the activation function, weight, and bias associated with the decoder layer.
To obtain satisfactory performance using the autoencoder scheme, the decoding loss should be minimized for which loss functions like SSEs (Sum Squared Errors) or RMSE (Root Mean Squared Errors) can be used.
In this paper, the stacked autoencoder was used on the input data to diminish the noise from the input data. Stacked autoencoder is basically a multilayered MLP, where output of each hidden layer is connected to the input of the successive layer. These layers are added to make sure the autoencoder does not just copy (the input) as (decoded output).
The architecture scheme is illustrated in Figure 4. It contains 1 input layer (x), and 3 hidden layers (1 corresponding to the encoder layer (W), 2nd corresponding to the output of encoder (z) and 3rd corresponding to the decoder layer (W')) and one output layer (x').

Now, the decoded inputs from this autoencoder are passed into our designed DNN and used for

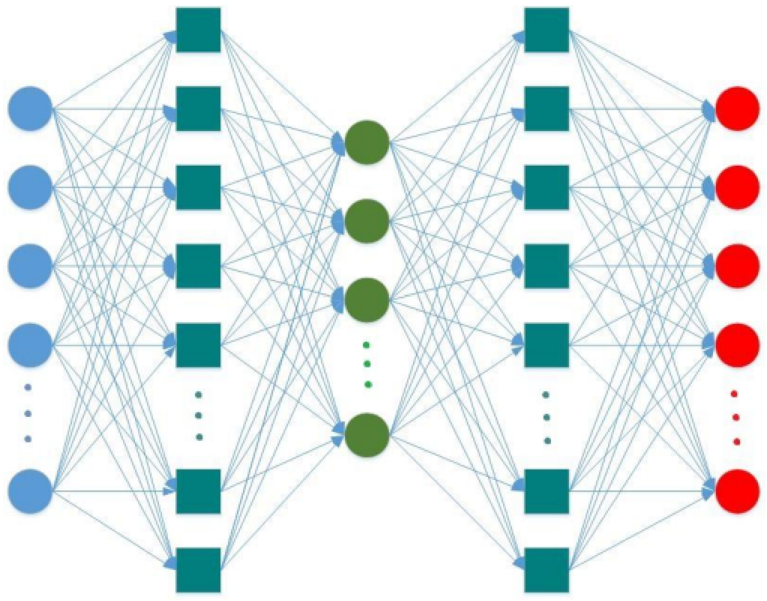


Figure 5: Structure of the autoencoder applied

training using the same settings as above. The results from the training and validation errors are illustrated in Figure 6 and Figure 7. As can be seen from Figure 6, the training errors did not increase as epoch number increased. However, validation errors became higher when the epoch number increased, as illustrated in Figure 7. Through the results, it is clearly seen from the graphs that the model suffered from an underfitting problem (because of the high training and validation errors).

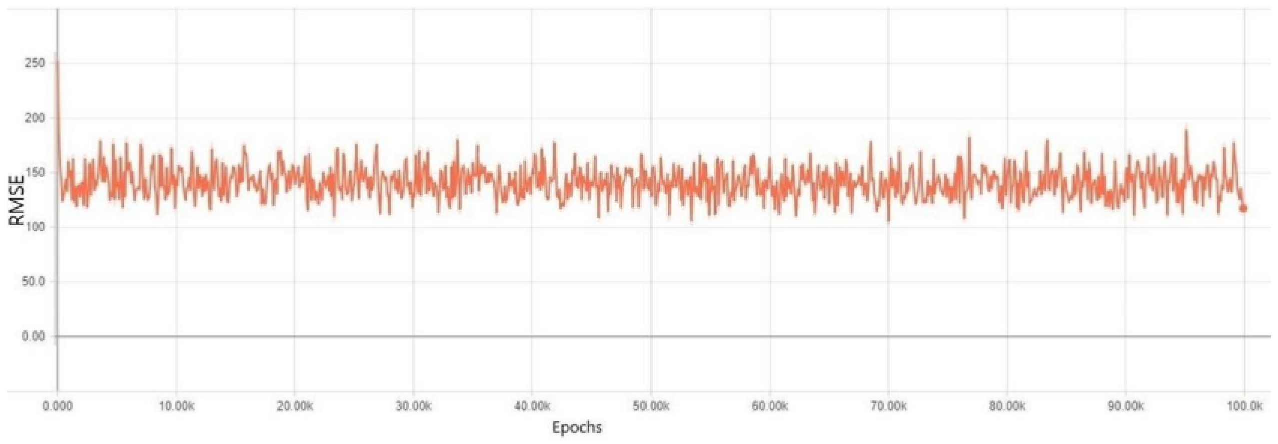


Figure 6: Training error with autoencoder applied

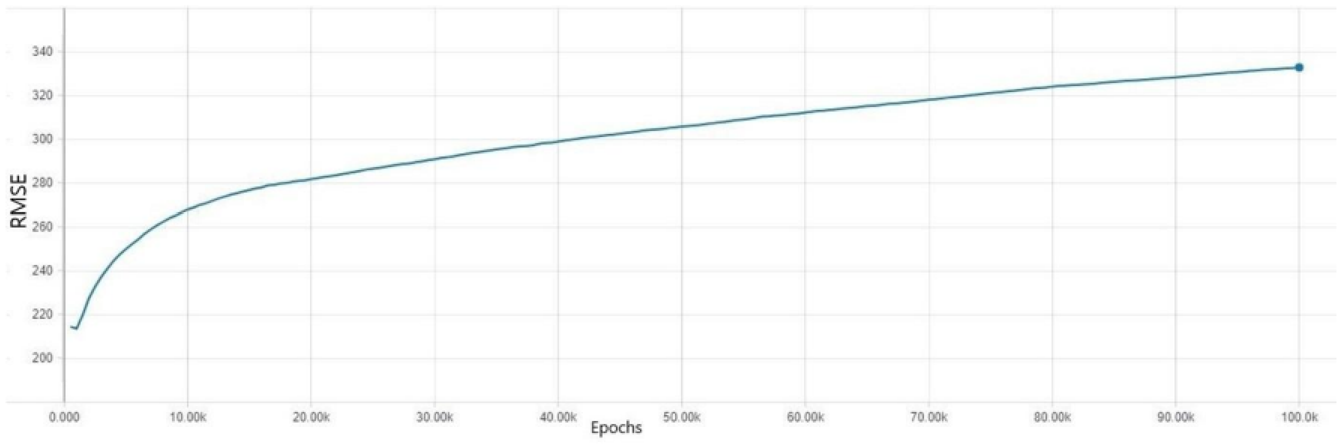


Figure 7: Validation error with autoencoder applied

7.3 Batch Normalization

Training DNNs with many hidden layers is challenging. One aspect of this challenge is that the model is updated layer-by-layer backward from the output to the input using an estimate of error that assumes the weights in the layers prior to the current layer are fixed.

Because all layers are changed during an update, the update procedure is forever chasing a moving target.

For example, the weights of a layer are updated given an expectation that the prior layer outputs values with a given distribution. This distribution is likely changed after the weights of the prior layer are updated. This change in the distribution of inputs during training is referred to as ‘Internal Covariate

Shift'. Batch normalization is proposed as a technique to tackle this problem by coordinating the update of multiple layers.
Again, the exact implementation of Batch Normalization is not provided in the paper, so we are guessing they just standardized the layer inputs.
For example- They standardized the layer inputs by calculating the mean and standard deviation of each input variable to a layer per mini-match and using these to perform the standardization (i.e. rescaling data to have 0 mean and a standard deviation of 1). It does this by scaling the output of the layer, specifically by standardizing the activations of each input variable per mini-batch, such as the activations of a node from the previous layer.
Standardizing the activations of the prior layer means that assumptions the subsequent layer makes about the spread and distribution of inputs during the weight update will not change, at least not dramatically. This has the effect of stabilizing and speeding-up the training process of deep neural networks. Normalizing the inputs to the layer has an effect on the training of the model, dramatically reducing the number of epochs required. It can also have a regularizing effect, reducing generalization error much like the use of activation regularization.
Now, coming back to training and validation RMSE errors obtained via this, we see fairly good results. The results of this technique are given in Figure 8 and Figure 9. As it is shown in the figures, the results were more acceptable than those using the autoencoder. The training errors began to decrease initially. However, the overall trend fluctuated constantly after approximately 3000 epochs. Validation errors decreased and increased from the beginning. Even though there was a small decrease around 10,000 epochs, the overall trend increased slightly. By comparing these two graphs, it became clear that the DNN model using batch normalization was overfitted within a small range, because validation errors increased in spite of the constant fluctuation in training errors.

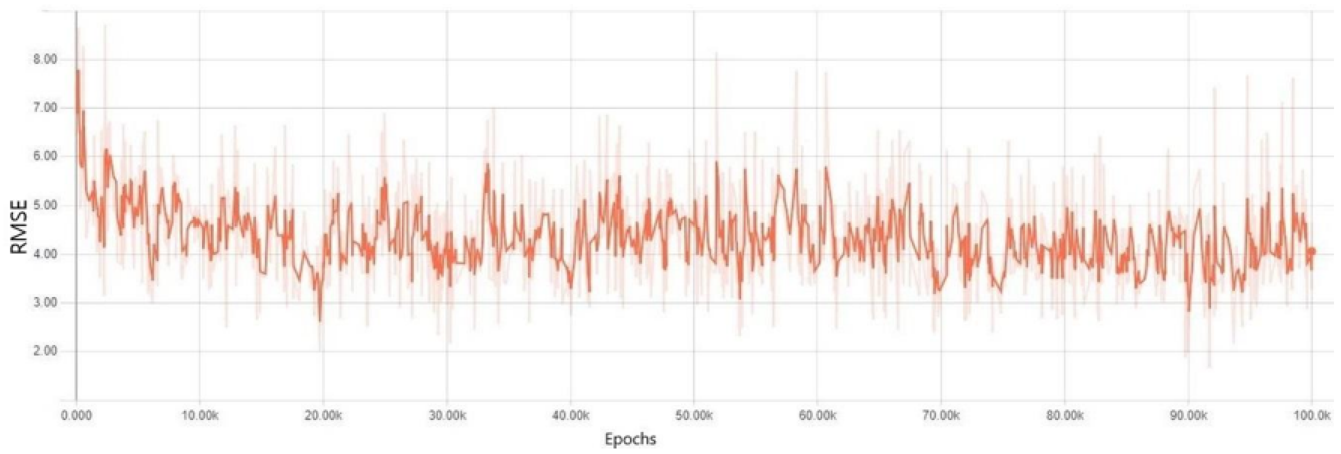


Figure 8: Training error with batch normalization applied

7.4 L1 Regularization

The next scheme applied in the experiment was the L1 regularization. The main idea behind the scheme is to regularize the loss function by completely removing the irrelevant features from the

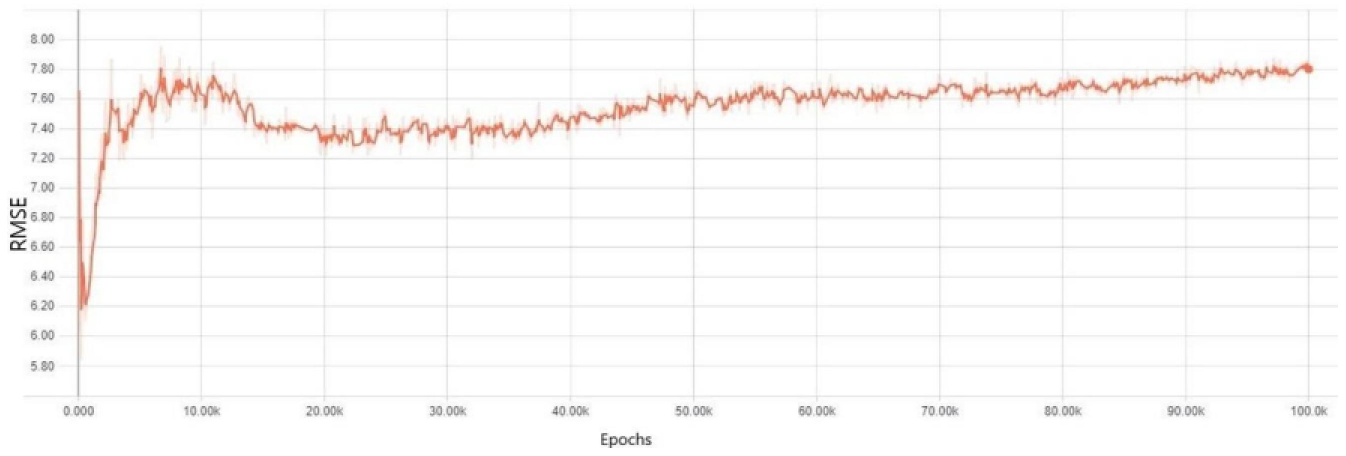


Figure 9: Validation error with batch normalization applied

model. It does so by reducing the sensitivity of the weights corresponding to irrelevant features (by reducing its magnitude). The equation of the scheme could be expressed as:

$$f(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\bar{y}_i, y_i) - \sum_{i=1}^{m} |w_j|$$

Here, $L(\bar{y}_i, y_i)$ is a loss function, m is the number of observations, $\bar{y}_i$ is the predicted value (whereas y_i is the actual value), and λ is a non-negative regularization parameter. The main objective was to minimize the $f(w,b)$ function by penalizing weights in proportion to the sum of their absolute values. As λ increases, w decreases.

Now, the experiment results are represented in Figure 10 and Figure 11. Figure 10 shows that the training errors were smoothly diminished as the epoch number increased. However, for validation errors, the trend showed a rise from the beginning of the epochs. This demonstrates that even though the L1 regularization technique was the most popular model to prevent overfitting in artificial intelligence, it still suffered from an overfitting problem (because of the low training, but high validation errors).

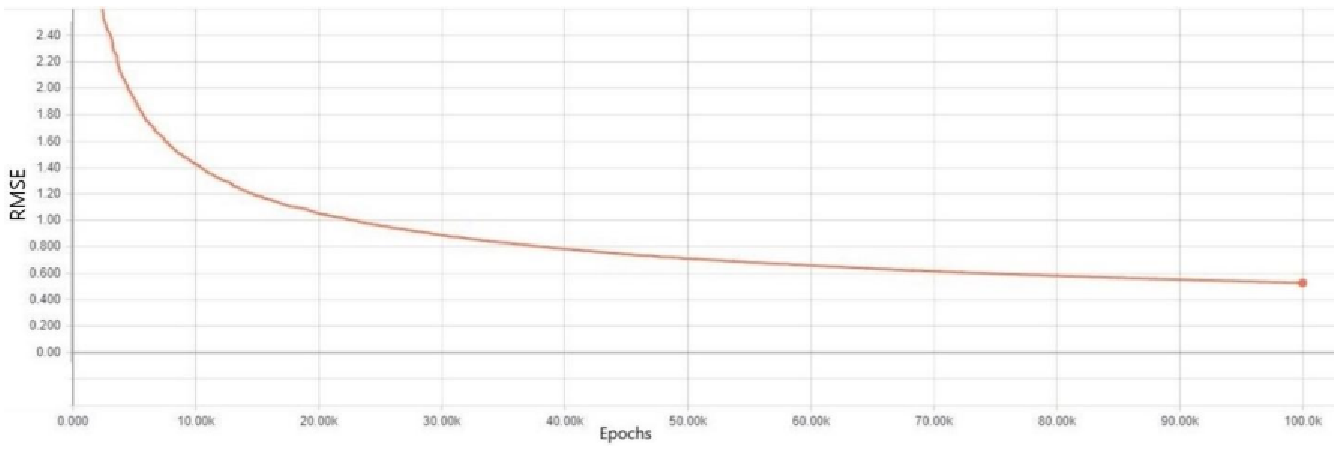


Figure 10: Training error with L1 regularization applied

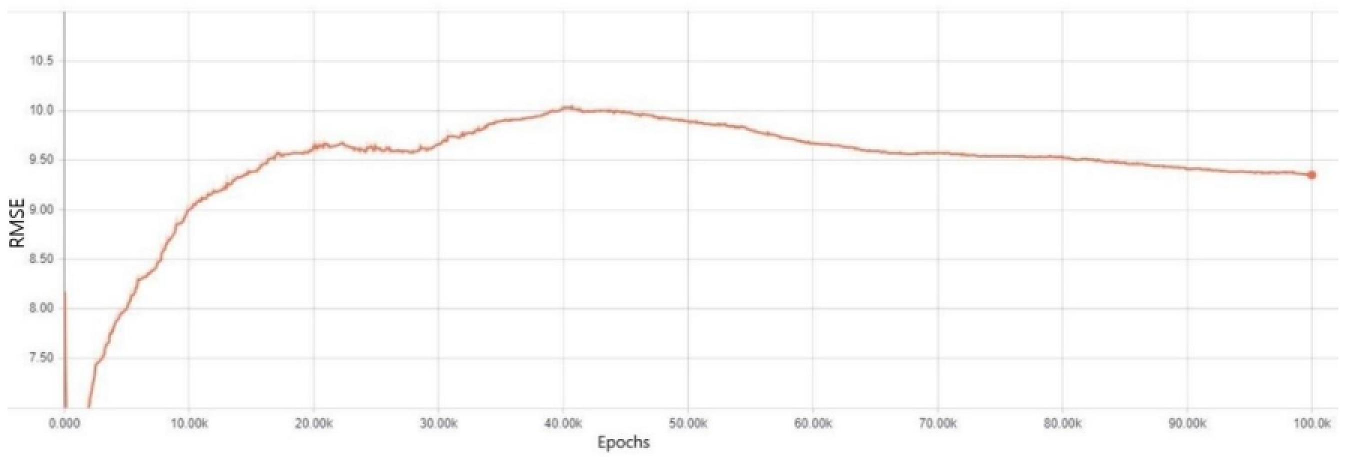


Figure 11: Validation error with L1 regularization applied

7.5 Data Augmentation

Data augmentation is one of the most popular regularization techniques. The main idea of the scheme is to expand the training dataset by applying transformations to decrease overfitting. This technique is commonly used in image processing, since image operations like rotating, shifting, scaling, mirroring, or randomly cropping can be easily implemented when using the scheme.
In this study, 2 simple augmentation techniques were used.
Here, instead of all 35 dimensions for 7 features over 5 consecutive days, we focus on a 7 dimensional vector, where each entry corresponds to a feature representative of all 5 days, either by considering the average of individual features over 5 days or by just considering the sum.
In both these techniques, the number of input neurons become 7. Hence, the number of operations in the model decreased as the number of input data decreased, thereby helping in preventing overfitting.

1. The first type of data augmentation was to just sum up the partial datasets (without taking their average like the second case). Suppose, $X_j^{(i)}$ represents the input feature data with 35 dimensions (for each $i \in [5]$ and $j \in [7]$). If the augmented input is z^k for each $k \in [7]$, we have:

$$z_k = \sum_{i=1}^{5} X_k^i$$

   The first scheme, which summed the features, performed better than other regularization methods. As shown in Figure 12 and Figure 13, the training errors simultaneously declined as the number of epochs increased. The validation error of the model found its optimal value at 40K epochs. The validation data error rose slightly after 40K epochs. From these graphs, it is shown that the DNN model with data augmentation based on summing had a slight overfitting after 40K epochs.

2. The first type of data augmentation was to sum up the partial datasets and take their average. Suppose, $X_j^{(i)}$ represents the input feature data with 35 dimensions (for each $i \in [5]$ and $j \in [7]$).

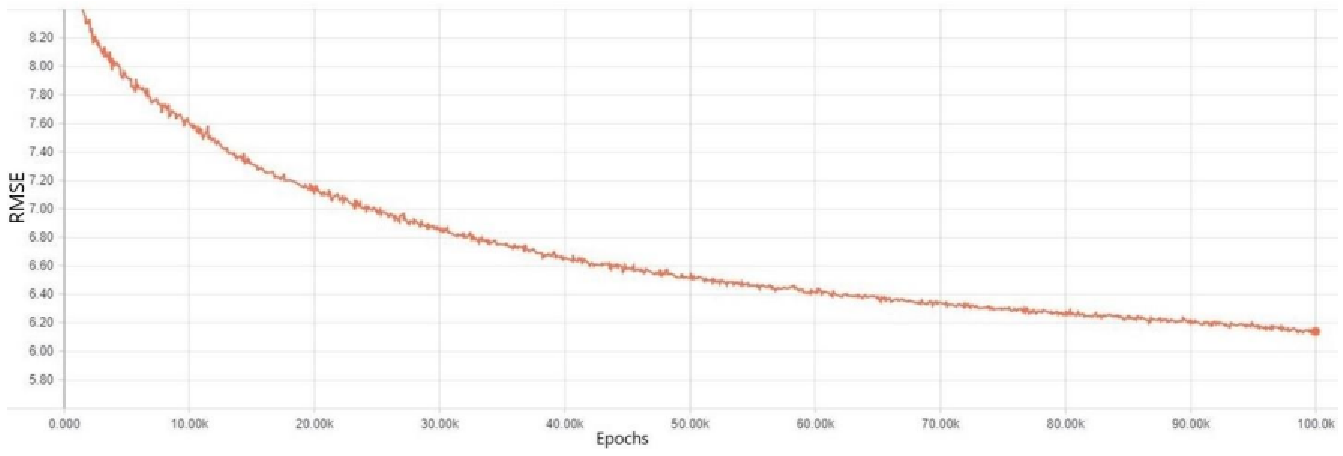


Figure 12: Training error with data augmentation applied

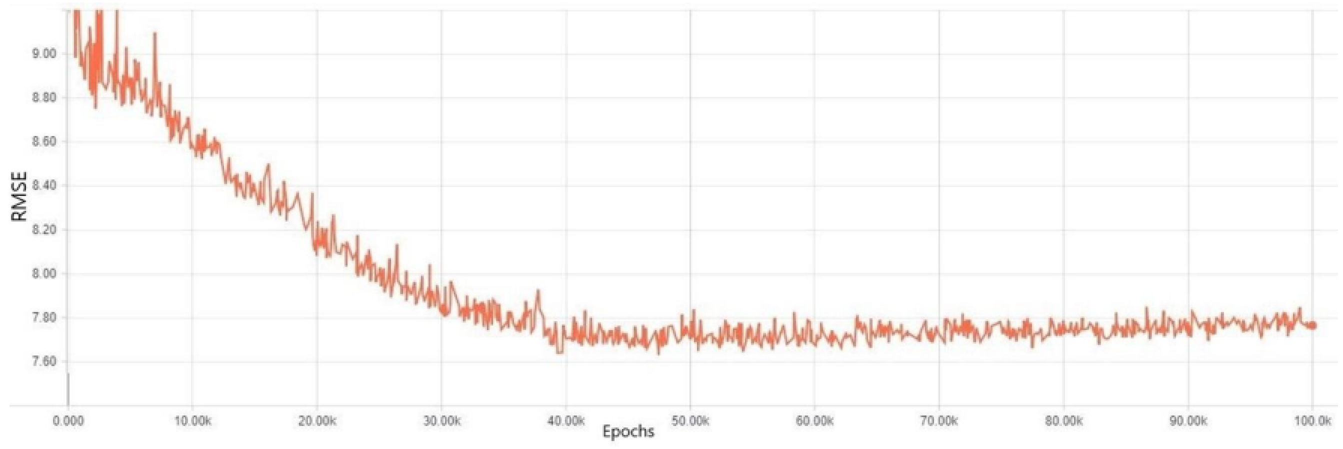


Figure 13: Validation error with data augmentation applied

If the augmented input is z^k for each $k \in [7]$, we have:

$$z_k = \frac{1}{5}\sum_{i=1}^{5} X_k^i$$

The experiment results using the data augmentation technique based on an average are illustrated in Figure 14 and Figure 15. The figures show that the overfitting issue was completely overcome with this method. As is shown in Figure 14, training errors were considerably diminished throughout all the epochs. For the validation errors, those rapidly decreased until 20K epochs and stayed stable after the point shown in Figure 15.. Then, the validation errors began to fall down slowly from 35K epochs, and it showed the smallest error at 100K epochs. Notice the difference between training and validation errors were not high. This indicates that the dataset achieved good generalization.

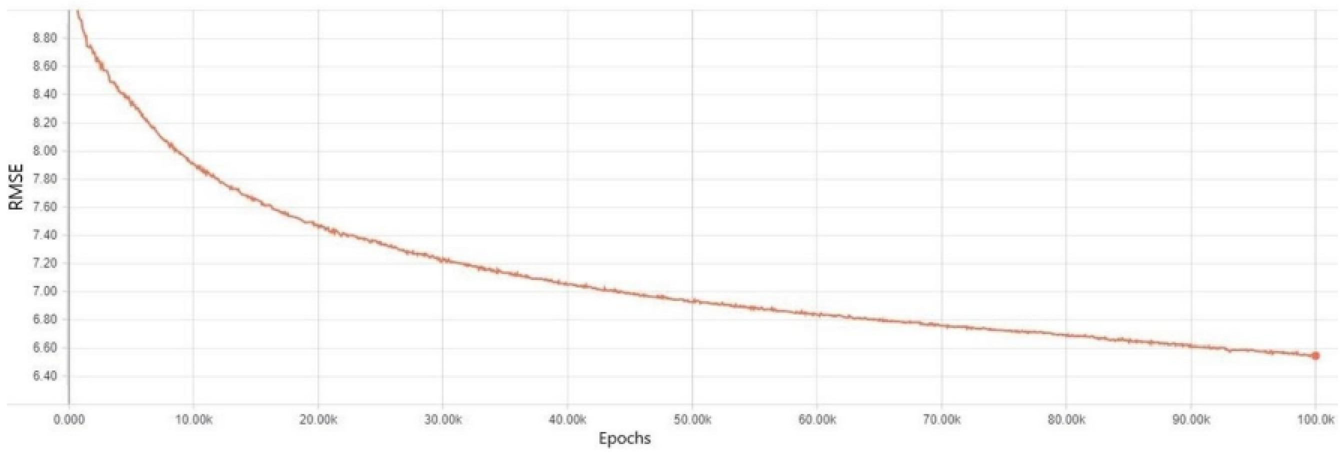


Figure 14: Training error with data augmentation applied, with average values of same categorical features taken

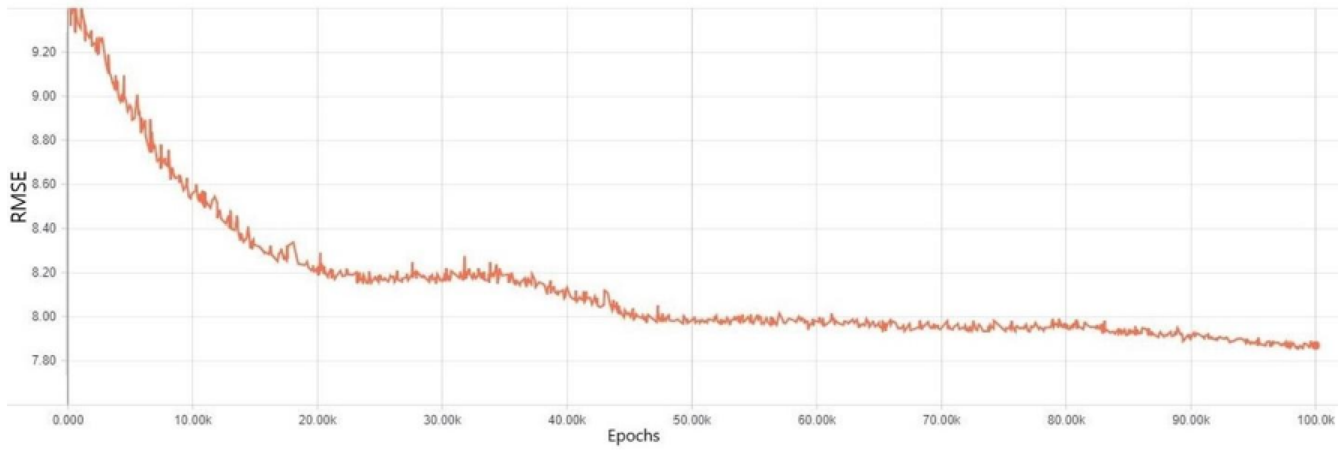


Figure 15: Validation error with data augmentation applied, with average values of same categorical features taken

8 Result

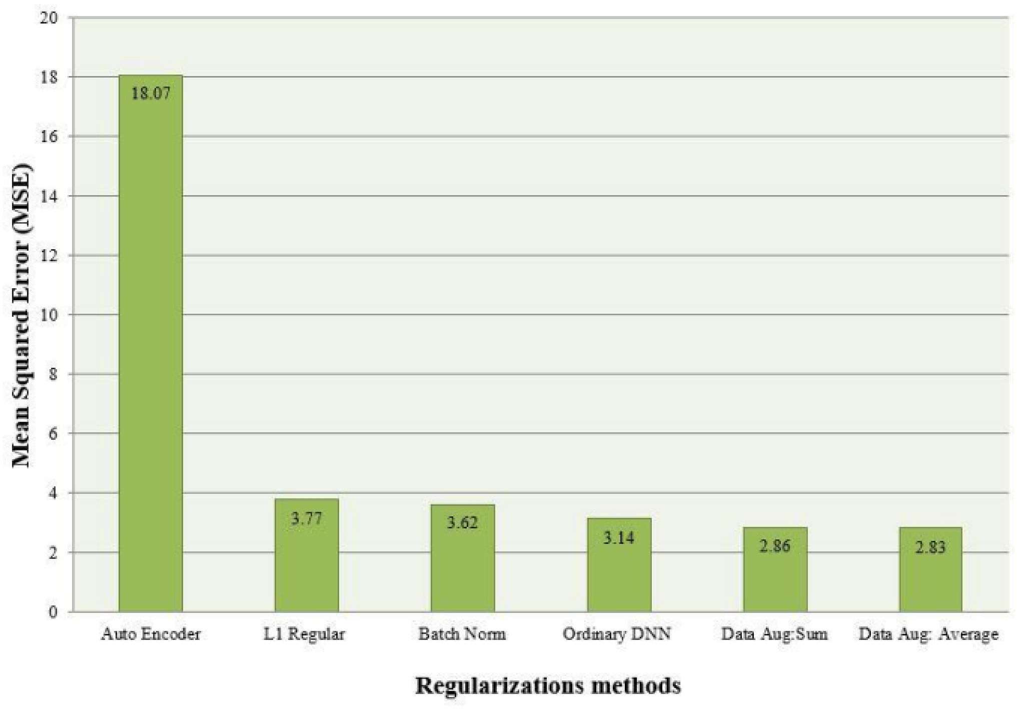
To evaluate the accuracies of each scheme we compare the averaged MSEs over the test set via the equation:

$$\mathrm{MSE} = \frac{1}{n}\sum_{k=1}^{n}\left(\overline{X}_k - X_k\right)^2$$

to judge the precision and accuracy of the predictors. The results are represented in the following graph.

As we can see, the value of error for autoencoder scheme was fairly high. For the other schemes, the values were not very high. L1 regularization and the autoencoder still encountered overfitting and underfitting problems. The batch normalization showed better performance than these methods, and the DNN model with data augmentation showed the best performance.

Finally, the paper compared the actual average temperature and predicted average temperature in all models. The prediction was done during ten days, from 2018.03.01 until 2018.03.10. The results are



shown in Table 3.

As can be seen from the table, the scheme that showed the worst performance was the autoencoder

Table 3. Comparison of actual and predicted average temperature.

Date	DNN without Regularization Methods (°C)	L1 Regularization (°C)	Autoencoder (°C)	Data August Sum (°C)	Data August: Average (°C)	Batch Normalization (°C)	Real Temperature (°C)
2018.03.01	2.3	6.3	10.1	1.5	1.0	3.9	4.6
2018.03.02	1.1	−1.9	17.6	−0.2	−0.8	1.0	−0.7
2018.03.03	2.9	−0.5	15.8	0.5	1.4	2.8	7.9
2018.03.04	3.5	17.1	15.1	1.9	0.6	10.3	9.8
2018.03.05	9.3	3.7	10.6	0.7	1.4	16.6	5.5
2018.03.06	4.3	2.7	17.7	4.8	4.5	−0.1	4.5
2018.03.07	2.6	4.4	17.6	8.3	9.1	9.4	6.4
2018.03.08	3.1	7.2	15.2	12.4	7.8	5.6	4.6
2018.03.09	6.6	3.1	16.6	2.7	4.7	4.0	4.5
2018.03.10	1.4	6.1	18.8	4.3	4.9	4.9	4.6

because there was a big difference between actuality and prediction. For data augmentation and batch normalization, the differences were fairly small. Sometimes, batch normalization outperformed data augmentation during some days. From the table, we see that the data augmentation showed the best performance because the prediction was nearly the same as the real temperature for some days.

9 Conclusion

The study showed that some models using regularization techniques demonstrated better performance than those without regularization methods in terms of training errors.

When comparing each scheme quantitatively, an autoencoder scheme exposed higher errors than other schemes. This was because it encountered underfitting caused by removing some of the training data. Therfore, it is concluded that the portion of removed data must be decreased for an autoencoder when the training data are insufficient. In addition, L1 regularization still encountered overfitting and underfitting. Batch normalization and data augmentation showed better performance than the others when comparing the errors.

When comparing the prediction accuracy, data augmentation and batch normalization still showed better performance than others. Of the two schemes, batch normalization outperformed data augmentation on some days. This was because much more training data was added to the original data instead of being removed. However, if too much data was used for training, it required too much time to complete the training of the models, demonstrating a tradeoff between training data and processing time.

The main contribution of this work is to help developers to choose the most suitable scheme for their neural network application by doing comparative research with the purpose of assessing the training and validation errors of a model with regularization methods.

From the study, we see that regularization methods could solve overfitting and underfitting problems efficiently, but, even though some regularization algorithms were applied, neural network models still suffered from the same problems during training. This indicates that it is not easy to solve the problems and a more enhanced solution needs to be devised to completely solve the problems.