Machine Learning for Software
Bug Categorization

Arman Vatandoust
Abstract

Machine Learning for Software Bug Categorization

Arman Vatandoust

The pursuit of flawless software is often an exhausting task for software developers. Code defects can range from soft issues to hard issues that lead to unforgiving consequences. DICE have their own system which automatically collects these defects which are grouped into buckets, however, this system suffers from the flaw of sometimes incorrectly grouping unrelated issues, and missing apparent duplicates. This time-consuming flaw puts excessive work for software developers and leads to wasted resources in the company. These flaws also impact the data quality of the system’s defects tracking datasets which turn into a never-ending vicious circle.

In this thesis, we investigate the method of measuring the similarity between reports in order to reduce incorrectly grouped issues and duplicate reports. Prototype models have been built for bug categorization and bucketing using convolutional neural networks. For each report, the prototype is able to provide developers with candidates of related issues with likelihood metric whether the issues are related. The similarity measurement is made in the representation phase of the neural networks, which we call the latent space. We also use Kullback–Leibler divergence in this space in order to get better similarity metrics.

The results show important findings and insights for further improvement in the future. In addition to this, we discuss methods and strategies for detecting outliers using Mahalanobis distance in order to prevent incorrectly grouped reports.
Acknowledgement

I would like to thank my supervisor Konrad Magnusson for his valuable guidance and support as well as Tom Olsson for his warm reception, help and guidance during my project. I also want to thank my reviewer Olle Gällmo for all the valuable advice, insights and important feedback during this period. The opportunity to pursue my Master’s thesis at EA DICE has been greatly appreciated and special for me and I would like to thank all fellow colleagues at EA DICE for their support and help.

Last but not least, I would like to thank my family for believing in me, supporting me and encouraging me to pursue my dreams.
# Table of Contents

1 Introduction ........................................................................ 1  
1.1 Objectives .................................................................... 2  

2 Background ........................................................................ 3  
2.1 Artificial Neural Networks .............................................. 3  
2.2 Convolutional Neural Networks ...................................... 5  
  2.2.1 Convolutional Layer .................................................. 7  
  2.2.2 Pooling Layer .......................................................... 8  
  2.2.3 Activation Layer ...................................................... 9  
  2.2.4 Fully Connected Layer .......................................... 12  
  2.2.5 Flatten layer ........................................................... 13  
  2.2.6 Dropout layer ......................................................... 13  
  2.2.7 BatchNormalization layer ..................................... 13  
2.3 Word Embeddings ........................................................ 15  
2.4 Character Embeddings .................................................... 15  
2.5 Mahalanobis distance ..................................................... 16  
2.6 Kullback–Leibler divergence ............................................ 17  
2.7 Optimizer .................................................................... 17  
  2.7.1 Adam ................................................................. 18  
  2.7.2 Nadam ............................................................... 18  
2.8 EarlyStopping .............................................................. 18  
2.9 Datasets ....................................................................... 19  

3 Dataset ................................................................................ 19
1 Introduction

As a games company DICE\(^1\) spends a significant amount of resources on tracking, fixing and regressing code defects in order to deliver an immersive and reliable gaming experience to their players. These defects range from soft issues such as usability or gameplay problems, to hard bugs such as crashes or hangs. These faults are automatically collected in their systems and grouped into clusters for tracking of similar faults. Clusters are collections of reports grouped together by a heuristic which ideally only contains reports which are caused by the same software bug.

It is not uncommon that crashes that are caused by the same bug land in different clusters. This is called ‘the second bucket problem’ [23][17]. Another problem which is ‘the long tail problem’ which occurs when there are only one or few reports in each bucket. Another flaw of the system is that it sometimes incorrectly groups unrelated issues, and misses apparent duplicates. These problems often lead to a misunderstanding of the severity of crashes and thus obscuring the importance of the crashes.

DICE has a significant amount of data which can be used when researching regarding how to find correlations and differences between different crashes and bugs. Being able to correctly group related issues and crashes together without duplicates will be an important step in their systems. This will also save DICE a significant amount of resources which can be spent on other important tasks.

In order to improve the existing crash system, the goal is to create a model which can determine the similarity between two reports and provide developers with some likelihood metric for whether these reports are related. To investigate this, we will investigate how we can use the ‘Latent Space’ in the model to measure the similarity between reports by using their call stacks. The latent space is a representation of each call stack input to the neural network models. Each representation is the result of the feature extraction part of the models (See figure 1).

\(^1\)https://www.dice.se
Measuring the similarity between different reports is a non-trivial task since the definition of similar reports can differ between different developers.

Machine learning approaches will be used to investigate this problem further. More specifically, different deep neural network architectures will be created, compared and evaluated in order to find the model which is best suited for this problem.

1.1 Objectives

The objective of this thesis is to investigate how neural networks can be used to determine the similarity between call stacks (reports). The investigation works as follows:

1. Build and train different neural networks to predict the correct bucket for each report.

2. Investigate the last dense layer of different neural networks which we will call the Latent Space. The idea is that similar inputs to different neural networks build similar representation in the network and hence, a similarity metric could be created by using Euclidean distance between different inputs in the Latent Space. In the case where similar reports build cluster groups in the Latent Space, a more powerful distance metric will be used, namely, Mahalanobis distance [50] which has its advantages over Euclidean distance (see 2.5). The idea of that similar classes should produce similar representation inside the network is mentioned by Simon S. Haykin in [25][26].

3. To make the Latent Space more friendly for distance measurement, we will implement a custom loss which will make the Latent Space normal distributed by using KL-divergence [45] [44]. We will do this by measuring the output of the Latent Space while training the neural network. The neural network must learn to predict the right bucket for any given report while maintaining a normally distributed Latent Space.
4. Some of the reports in the database with different buckets have been merged to the same bucket by developers because of similar or identical reports. We will use this information from the database to help our neural networks to approximate the right function.

5. In addition to this, we will also introduce Jira classes. Jira classes contain groups of buckets which in turn contains one or more reports. Jira classes are created by developers which group similar buckets. These groups will be called Superclusters (See figure 2. This information is also fetched from the database.

6. Evaluate the models which have performed best on test accuracy and investigate whether the output from the Latent Space is suitable for measuring the similarity between call stacks.

2 Background

2.1 Artificial Neural Networks

The brain has the ability to perform a significant amount of tasks such as motor control, image recognition, pattern recognition, processing of sensory information and many more. It can do all this thanks to biological neurons (BN). These neurons are arranged in modules which build up neural networks. Although there is no actual exact estimate of the number of biological neurons in the human brain, different authors have estimated the total number of neurons to be around 85-120 billion.[79][4][28][27]. Neural
networks in the brain give humans the ability to learn, memorize and still generalize. These abilities are the main source which inspired Artificial Neural Networks.

An Artificial Neural Network (ANN) is a neural network consisting of Artificial Neurons (AN). These artificial neurons are modelled after biological neurons in the brain. ANs receives one or more input $x_1, x_2, x_3, \ldots x_n$ which are multiplied with coefficient $w_{ji}$. These coefficient are called weights, which together with the inputs builds linear combination which is the calculation of the input to a summation function which outputs $y$. The output of $y$ can be described with the following equation:

$$y = \sum_{i=1}^{N} x_i w_i + \theta$$  \hspace{1cm} (1)

Where $\theta$ is an extra input with a weight value of 1 which represent the threshold of the neuron, sometimes also called bias of a neuron. The value of $y$ is then passed into a second function, an activation function which generates the final output $Y$ from the neuron. There exist several activation functions which define the characteristics of the neuron. The idea of an artificial neuron was first described by Warren McCulloch and Walter Pitts in 1943 [51]. Figure 3 shows an illustration of the artificial neuron.

ANNs consist of one or more layers containing ANs, also called hidden nodes or hidden units. One common setup of ANN consists of an input layer, one hidden layer and one output layer which can be observed in figure 4. The ANN can be configured for different applications such as data classification, image recognition, face recognition, speech recognition, data prediction and many more. They are able to do this through a learning process which involves adjustments of the weights in the ANN. This process is inspired by our biological neurons which learn through adjustments in the synaptic connections.
Neural networks have different architecture types, two of them are Convolutional Neural Networks and Recurrent Neural Networks which are mostly used for classification problems.

## 2.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) is another type of Neural Networks which is widely used in computer vision tasks. They have traditionally been used mostly in applications in image and video recognition, image classification and medical image analysis. But in recent years, they have started to be used in natural language processing as well, where they have achieved state-of-the-art results\[81\] [14].

In 2012, Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton made a breakthrough when they introduced the first convolutional neural network architecture (named AlexNet) that achieved strong performance on image classification. They created and trained their deep model on 1.2 million images to classify 1000 unique classes. With their architecture, they achieved a winning top-5 test error rate of 15.3%, compared to 26.2% achieved by the second-best entry in the ILSVRC-2012 competition. [43]

Their work was an important factor in accelerating the field of convolutional neural networks and allowed the community to research new architectures, constantly beating the state-of-of-the-art results in the field. Today convolutional networks are widely used in several different applications.

The inspiration of convolutional neural networks can be traced back to 1959 where
David H. Hubel and Torsten Wiesel[29] did experiments on the cat’s visual cortex in order to learn more about how the neurons in the visual cortex work. They did a series of experiments where they provided cats with different stimuli in the form of different shapes and they measured the response of the neurons in the cat’s brain from the stimuli. Measurements were made with electrical signals from the cat’s brain. [29] [34] [30] [31] [32] [33].

In 1980, Kunihiko Fukushima presented the Neocognitron[22] which showed the first example of a neural network model which had the architecture of the idea of simple and complex cells which Hubel and Wiesel had researched on [22]. Fukushima represented these simple and complex cells in layers.

The first example of a convolutional neural network using backpropagation was presented in 1998 by Y LeCun, L Bottou, Y Bengio and P Haffner who used CNNs for handwritten character recognition. At this time, they compared convolutional neural networks to other methods and showed that convolutional neural networks outperformed all other methods. LeNet-5 performed well on digit inputs but were limited in terms of scalability and hence couldn’t handle more complex input data than digits.

AlexNet[43] was similar to LeNet but were scaled to be deeper, larger and more complex. A rich amount of data and the possibility of parallel computation with multiple GPUs allowed AlexNet to show the true potential of convolutional neural networks and hence, open new opportunities in the deep learning community.

Convolutional neural networks have shown excellent performance in various kinds of problems in computer vision. For example image classification/recognition where there is an image input and the goal is to determine which class the input belongs to, which could be a cat or dog. This is also called a binary classification problem. Convolutional neural networks can also be used in cars for object detection, for example with the purpose of detecting other cars and hence, avoiding collisions with them. In recent years, convolutional neural networks have also shown to perform well in natural language processing.

One of the challenges in computer vision is dealing with large images which leads to big input vectors. For example an image input with dimension 2000x2000 gets multiplied with 3 because of RBG channels and that leads to an input vector with dimension 12 million. Taking this input feature in an artificial neural network (shown in 4) with a single hidden layer of 512 neurons would result in 512x3M weights and a total of 6144M parameters (6.144 billion). This number of parameters would be impossible to train due to the computation and memory requirements but also training such neural network would very easily overfit to the training data.

To overcome this problem one can use the convolutional operation which is one of the

2.2.1 Convolutional Layer

The purpose of the convolutional layer is to take an input (in forms of an image, audio or text) and convolve with an arbitrary filter (also called kernel) and a given stride in order to get an output with reduced dimensionality. The number of strides determines how many steps the filter moves, for example, with an image input, a stride of 1 would mean that the filter moves after each pixel. Larger stride means less overlapping between the pixels in the input given that the filter and stride fits with the input. An input with the size I 6x6 and a filter with size F of 3x3 would get the output size of 2x2. See figure 5.

When handling inputs like images, the filter is convolved with the image spatially to preserve spatial structure so the filters always extend the full depth of the input volume. For example, an input with RBG channel would have a filter of 3x3x3 since there are 3 channels in depth.

It is common to have several convolutional layers and also multiple filters in a convolutional neural network. This way, the dimension is reduced after each convolutional with each filter.

A convolutional is a mathematical operation which takes two functions and produces a third function by taking the dot product of each cell in the filter with the input. In this case, the first function would be the input \( i[x] \), the second function would be the filter/kernel \( k[x] \) and the third function would be the result, also called feature map or activation map \( y[x] \). The definition of a 1d convolutional with discrete variables is:

\[
y[x] = i[x] * k[x] = \sum_{-\infty}^{\infty} i[m] \cdot k[x - m]
\]  

[16]

The equation for 2d convolutional with discrete variables is:

\[
y[x, y] = i[x, y] * k[x, y] = \sum_{-\infty}^{\infty} \sum_{-\infty}^{\infty} i[n_0, n_1] \cdot k[x - n_0, y - n_1]
\]  

The output of a convolutional layer can be given with the equations above.

For simplicity, we can show an example. Given an image in greyscale, we have a 6x6 matrix which represents the input image. See figure 5. To get the value of the result
2 Background

matrix for the first cell (upper left corner) we can use the equation above:

\[ y[x, y] = i[0, 0] \ast k[0, 0] + i[0, 1] \ast k[0, 1] + i[0, 2] \ast k[0, 2] + \\
i[1, 0] \ast k[1, 0] + i[1, 1] \ast k[1, 1] + i[1, 2] \ast k[1, 2] + \\
i[2, 0] \ast k[2, 0] + i[2, 1] \ast k[2, 1] + i[2, 2] \ast k[2, 2] = \\
1 \ast 1 + 2 \ast 0 + 9 \ast 1 + 2 \ast 0 + 4 \ast 1 + 1 \ast 0 + 5 \ast 1 + 2 \ast 1 + 4 \ast 1 = 25 \] (4)

A convolutional neural network architecture consist of several blocks such as fully connected layer, convolutional layer and pooling layer.

Figure 5: Convolutional operation on a 6x6 input with a 3x3 filter and stride of 3 which produces a feature map of size 2x2

2.2.2 Pooling Layer

Pooling layers are also one of the essential components when building convolutional neural networks and is often applied after the convolutional layer. The purpose of pooling layers is to reduce the spatial dimension of the feature map (activation map). By reducing the dimension, the pooling layer reduces the number of parameters and hence, reduces the computation required to train the model. Pooling layers differs from convolutional layers in that they are not trainable.

There exist different kind of pooling layers, for example, average pooling layer and the more common max pooling layer which have shown to be more practical and useful to use as a pooling layer[68]. Similar to convolutional layers, pooling layers also have a kernel (filter) and stride which are used to reduce the size of the feature map. For example, having an input of size 6x6 produces a 3x3 output with an 2x2 max pooling layer (See figure 6).
As seen in figure 6, the max pooling layer operates with a stride of two, each time taking the highest value of the input feature map to the output feature map, hence, the output feature map is a downsampled version of the input feature map. Another property that makes max pooling layer powerful is that it has no trainable parameters, hence, a fixed computation.

### 2.2.3 Activation Layer

The activation layer introduces non-linearity to allow the network to learn more complex functions. It takes input and applies a function to some output seen in figure 3. There exists many different activation functions, each with unique characteristics which can be used to allow the neural network to learn various complex functions. Some of the more common activation functions in convolutional neural networks are:

**Logistic function**

The sigmoid function is a common activation function used in the final output layer of convolutional neural networks for binary classification problems. It maps the input between 0 and 1 and be written and illustrated as:
The TanH function maps the input between -1 and 1 and can be written and illustrated as:

\[ f(x) = \frac{1}{1+e^{-x}} \]

\[ y = \tanh x \]

**Rectified Linear Unit function, ReLU**

The ReLU activation function is one of the more common activation function in neural networks. It has gained its popularity during recent years, having the advantage of faster training time than the hyperbolic tangent (TanH) function [43] [82]. But also the advantage of better performance [49].

The ReLU activation function maps negative inputs as 0 and positive inputs remain as it is, so it works by removing negative inputs. Removing negative features in the input
can cause information loss [82] since negative features can contain information. This can be a possible disadvantage and therefore, we will also experiment with LeakyReLu described below ReLu figure. ReLu function can be written and illustrated as:

\[ y = \max(0, x) \]

**LeakyReLU**

The LeakyReLU activation function is similar to ReLU but doesn’t remove negative inputs entirely, but rather reduce the magnitude of negative inputs by a parameter \( \alpha \). This can be seen as an improved version of the ReLU activation function since it preserves negative features in the input. Here, with \( \alpha = 0.3 \), the LeakyReLU function can be written and illustrated as:

\[ y = \max(\alpha x, x) \]

**Softmax activation function**
2 Background

The softmax activation function is commonly used in the final layer of a neural network and is often used as the output in multi-label classification to determine the probability of the output class. For example in a multi-label classification problem with five classes, the output of the Softmax layer would be a vector with size 5x1 (see figure 7) where the sum of all probabilities for each class would sum up to one, that is:

\[ P(y = 1|x) + P(y = 2|x) + P(y = 3|x) + P(y = 4|x) + P(y = i|x) = 1 \]

Figure 7: Softmax layer with five classes as output

The probability for the \( i \)'th class given a vector \( X \) with weight vector \( W \) is given by:

\[
p(C_i|x) = y_i(x) = \frac{\exp(w_i^T x)}{\sum_k \exp(w_k^T x)}, \quad k = 1, \ldots, K \tag{5}
\]

2.2.4 Fully Connected Layer

The fully connected layer, also called a dense layer, is one of the most common layers in neural networks. It is common to have some fully connected layers in the end of
neural networks. [8] [47] [43] [81] [70] [75] [13]. In a fully connected layer, all the nodes, i.e. the neurons, are connected to the previous layer. They are used for calculating the weighted sum on all the extracted features of previous layers. The fully connected layers hold the processed information from previous layers and work as a classification module in the convolutional neural network. A fully connected layer is said to be fully connected when every node in that layer is connected to every other node in adjacent forward layer [25] [56] [35] [19]. See figure 7.

2.2.5 Flatten layer

When designing convolutional neural networks, it is common to have some fully connected layers before the output layer. In order to transform the multidimensional output from previous convolutional/pooling layers into one dimensional tensor (vector) for the fully connected layer, the flatten layer can be used. For example, if the output from the previous layer is of shape \((4, 5, 64)\) then the output from the flatten layer would be a one dimensional vector of \((4 \times 5 \times 64 = 1280)\). This one-dimensional output would then be compatible with a fully connected layer.

2.2.6 Dropout layer

Dropout is a regularisation technique presented in [71]. The technique extends the idea of adding noise to the input in the context of Denoising Autoencoders where the network is trained to construct the noise-free input [76] [77]. The idea is to randomly turn off neurons (hidden nodes) in the network based on a given drop rate. For example, a drop rate of 0.2 would mean that there would be a 20\% chance of dropping a random hidden node. This forces the neural network to learn redundant representation, hence, making the trained neural network more robust and more resistant to overfitting. Dropout also helps in terms of performance due to the fact that dropping hidden nodes also means fewer connections and therefore, fewer weights to be calculated during training time.

2.2.7 BatchNormalization layer

Batch normalization is a technique presented in [36] which significantly increases training speed and stability of a neural network by reducing internal covariate shift. This is defined as the change in the distribution of network activations due to the change in network parameters during training. The authors address the problem by normalizing layer inputs as follows:
1. Calculate mini-batch mean:
\[
\mu_\beta \leftarrow \frac{1}{m} \sum_{i=1}^{m} x_i
\]

2. Calculate mini-batch variance:
\[
\sigma^2_\beta \leftarrow \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_\beta)^2
\]

3. Normalize the batch normalization layer input:
\[
\hat{x}_i \leftarrow \frac{x_i - \mu_\beta}{\sqrt{\sigma^2_\beta + \epsilon}}
\]

4. Scale and shift the input which will be the output for the next layer:
\[
y_i \leftarrow \gamma \hat{x}_i + \beta
\]

where \(\gamma\) and \(\beta\) are trainable parameters.

In addition to faster training speed and stability, Batch Normalization has also shown that it can act as a regularizer and in some cases eliminate the need for Dropout [36].

The effectiveness of Batch Normalization has been discussed in [67] where they have shown that the performance of Batch Normalization is not as effective as it is widely believed and that internal covariate shift is not a good predictor of training performance. Instead, they identify another key effect that Batch Normalization has on training process which is that the gradients during training are more predictive and well-behaved. This, in turn, allows for faster training and more effective optimization.

A recent paper shows that Batch Normalization can provably accelerate optimization by splitting the optimization task into optimizing length and direction of the parameters separately [42].

Batch normalization has shown to work in convolutional neural networks with state-of-the-art performance on several text classification tasks without the need of using dropout technique [14]. Other recent papers have also shown that Batch Normalization reduces covariate shift of convolutional neural networks [11]. Further, in [58], the authors have shown that batch normalization in convolutional neural networks requires less training time as well as less number of parameters compared to existing compared models. In addition to convolutional neural networks, batch normalization has also shown benefits
of improved optimization, faster training and better generalization in recurrent neural networks [15] [38].

Overall, batch normalisation is a powerful technique that facilitates the learning process and hence, leads to better convergence in deep neural networks [24].

2.3 Word Embeddings

Neural networks cannot handle raw text as input but must instead rely on numeric input. When working with text data, the data has to be converted to numerical input to be compatible with deep learning models. One technique for doing this is by using Word Embeddings.

Word embeddings are commonly used in natural language processing area to convert sentences, words or even characters to vectors with numerical values. The idea is that words with similar meaning should have similar representation. This idea can be traced back to 1957 [21]. There exist several methods to create and learn word embeddings, one which includes neural networks. One common way to obtain word embeddings is by having an Embedding layer as part of the neural network architecture and learn the word embedding space simultaneously while training your network. This is the way which we will use in this project. An advantage of this approach is that word embedding space gets specialised for your specific task, a downside is, however, an increased amount of trainable weights. Another common way to obtain word embeddings is by using pre-trained word embeddings. Some of famous pre-trained word embeddings include Word2Vec [57], Glove [61] and Doc2Vec [46].

2.4 Character Embeddings

Character embeddings work in the same way as word embedding expect that the numerical representation is built by characters instead of words. We will use all the characters used in English vocabulary and characters used in C# and C++ since that’s what the call stacks consist of. Unknown characters will be treated as a special character. The legal characters will, therefore, consist of 73 characters and they are:

```
abcdefghijklmnopqrstuvwxyz0123456789
-,.!?:’’’/\|_@#$%ˆ&˜+-=<>()[]{}ˆ>>
```

15
2 Background

The input to our model will be 3D tensors with shape (50,128,73). 50 stands for the number of rows from each call stack and 128 stands for the maximum length of each row, i.e. each function name. The depth of the tensor will be the embedding width which will be 73.

![Figure 8: Illustration of the input to our models. Each (50,128,73) matrix corresponds to one call stack from one report.](image)

2.5 Mahalanobis distance

Mahalanobis distance is a generalized distance measurement developed by Prasanta .C Mahalanobis in 1936 [50]. It is a multivariate distance measure which is described to be dimensionless [41], used to measure the distance between an object $x_i$ and the mean of the vector $\bar{x}$ [48] [37]. In our case, the object $x_i$ is the output vector from the latent space belonging to a call stack and $\bar{x}$ is the output vector in the latent space for inputs for each bucket of call stacks that is under the same super cluster class.

The Mahalanobis distance can be calculated by:

$$D^2(\bar{x}) = (\bar{x} - \mu_i)^T S^{-1}(\bar{x} - \mu_i)$$

where $\bar{x}$ is a matrix containing all the vectors from the latent space belonging to each
cluster class. For example a cluster class with 100 call stack inputs would have \( \bar{x} = [[0, 1, 0.5, \ldots, n], \ldots, k] \) where \( n = \) latent space dimension and \( k = 100^{th} \) vector.

\( \mu_i \) is the mean values of the cluster class matrix \( \bar{x} \) and \( S \) is the cluster class covariance matrix. In our case, each cluster class has its own covariance matrix \( S \) and \( \mu_i \) which can be interpreted as the signature of the cluster class. Mahalanobis distance can also be interpreted as a measure of the distance between two populations [25]. In our case, it can be used to measure the distance between two different cluster classes and hence get some kind of similarity metric between those classes. It can also be useful for determining the similarity between a new call stack and existing cluster classes. In addition, Mahalanobis distance is known to be able to detect outliers in multivariate data. This task is not trivial, especially when there are multiple outliers present [78]. Among several similarity metrics between groups, Mahalanobis distance has shown to be most suitable in several applications [55].

The Mahalanobis distance reduces to Euclidean distance in the case when the covariance matrix is the identity matrix [25]. In this special case, it would mean that the inputs for a certain cluster class have mapped to the same point in the latent space, hence, the similarity in the latent space between two classes would be measured with the Euclidean distance instead. The difference between Mahalanobis distance and Euclidean distance is that Mahalanobis distance takes covariances/correlations between variables into account, hence making the Mahalanobis distance relevant for correlated features when elements of the covariance matrix \( \Sigma \) is not equal to zero [39].

2.6 Kullback–Leibler divergence

Introduced by Solomon Kullback and Richard Leibler in 1951 [45], the Kullback–Leibler divergence can be used as a similarity measurement between two probability distributions, measuring how much one distribution differs from another [44].

We will use Kullback–Leibler divergence measurement in our custom loss in order to form a normal distributed latent space.

2.7 Optimizer

Keras supports a various amount of different optimizers such as, Stochastic gradient descent (SGD) [65], RMSprop [73], Adagrad [20], Adadelta [80], Adam [64] [40], Adamax [40], and the more recent Nadam [18] [72]. In this thesis we will experiment with Adam optimizer and Nadam optimizer.
2.7.1 Adam

Adam optimizer is one of the most common optimizers used in neural networks. Adam is a method for efficient stochastic optimization introduced in [40]. It is designed to combine the advantages of two other commonly used optimizers, RMSprop [73] and Adagrad [20].

Experiments will be made with default settings written in [40] which is: \( \alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8} \).

\( \alpha \) is the stepsize, also called learning rate. \( \beta_1 \) and \( \beta_2 \) are exponential decay rates for the moment estimates and \( \epsilon \) is a so called fuzz factor.

2.7.2 Nadam

Nadam is introduced in [18] with the goal of using a faster and more powerful optimizer. Nadam optimizer is much like the Adam optimizer expect that it uses Nestorov momentum[72]. Nadam have shown advantages and improvement over Adam optimizer. One of the advantages is that it is using momentum, which is also mentioned and highlighted in [72].

Experiments with Nadam optimizer will also be with default settings. These settings include: \( \alpha = 0.002, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, \text{scheduledecay} = 0.004 \).

2.8 EarlyStopping

EarlyStopping is a callback used in Keras to monitor desired quantity. These quantities can for example be the loss or the accuracy of the model when training. It is used to stop training once the model have stopped improving. The method can be seen as a form of regularization to avoid overfitting [63] [62] [9].

We will use EarlyStopping with patience of 5 which is the number of epochs with no improvement after which training will be stopped. The optimal value for this differ between different problems and it is a value that have to be experimented with. When using EarlyStopping, having insufficient information regarding training examples can increase generalization error [10].
2.9 Datasets

Different kind of problems requires different kind of architecture when designing neural networks. Finding good parameters and hyper-parameters of a model is often a non-trivial and time-consuming task. Searching for these parameters is an iterative process that has a heavy impact on the performance of the model.

A common approach for finding these parameters is to divide the dataset into three subsets consisting of a training set, validation test (also called development set) and a test set. Each subset has its own purpose and it can briefly be described as:

**Training set** - The training set is used during training to tweak the parameters of the model and finding the most optimal weights. It is usually the biggest subset of the whole dataset and ranges from 50%-95% depending on the problem and number of samples in the whole dataset.

**Validation set** - The validation set (also dev set) is like training set used during training time. It’s purpose, unlike the training set is to tune the hyper-parameters for the model. It is also used to find the stopping point when training in order to avoid overfitting.

**Test set** - The test set is unique in the sense it’s not used during training time but rather during test time. The purpose is to evaluate how well the model performs on data which it has not trained on, i.e unseen data. The better generalized model, the higher chance of performing well on the test data.

3 Dataset

The dataset was originally fetched from an internal DICE system with REST API. This was done by writing a python script which fetched an arbitrary number of reports for each cluster. This procedure was not optimal when fetching a large number of reports since it would put excessive load on the internal system.

To avoid this, a copy of the system were created on a database which removed all other dependencies and hence, reports could be fetched independently without putting excessive load on the system. This improved workflow and made fetching the data faster. Python scripts were implemented to fetch desired data from the database.

The internal system collects a significant amount of information for each report. See table 1. Note that this information is synthetic.

The data used in this thesis are primarily data from call stacks from each report. An
3 Dataset

<table>
<thead>
<tr>
<th>Report information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster ID</td>
</tr>
<tr>
<td>ID</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>Time</td>
</tr>
</tbody>
</table>

Table 1: Example of report information

example of a call stack from a given example can be seen in table 2. These call stacks are used as an input to machine learning models to predict the right Cluster ID.

<table>
<thead>
<tr>
<th>Memory</th>
<th>Call stack (function names)</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000000k</td>
<td>jonSnow::longclaw</td>
<td>oldTown\theCitadel\aegonTargaryen.cpp (998)</td>
</tr>
<tr>
<td>0x0000000050</td>
<td>got::jonSnow</td>
<td>oldTown\theCitadel</td>
</tr>
<tr>
<td>0x000000049</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0x00000002</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0x00000001</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0x000000000</td>
<td>sevenKingdoms::got</td>
<td>oldTown\theCitadel</td>
</tr>
</tbody>
</table>

Table 2: Example of the call stack part of a report in the internal system, note that names from this call stack are synthetic.

The order of the call stacks starts from 0 up to k, meaning that jonSnow::longclaw is the most recent call stack.

3.1 Data fields

- Call stack - List of function names
- clusterId - Integer.

3.2 Preprocessing and feature engineering

The data that are getting fetched from the database are saved as a numpy file with the following format:
Machine learning models such as convolutional neural networks and recurrent neural networks achieve optimal results when the data is filtered and preprocessed into a high-quality dataset, crafted for the specific problem that one is trying to solve.

The definition of a high-quality dataset depends on the problem and the context, for example in sentiment analysis for movie review, one might want to remove special symbols to clean the data since they won’t contribute to predicting the right output. In our case, the data consist of call stacks from C++ and C# language and hence, special symbols will be important in this context. Another example is when using the common preprocessing method of converting text into lower case. The model wouldn’t be able to learn the difference between the company ‘Apple’ and the fruit ‘apple’. It would interpret both as ‘apple’. This preprocessing method hurts performance in some contexts but might be effective in others. In the case of handling with the dataset in table 3, it is suitable for converting all text to lower case since it isn’t important considering the context.

The dataset is prepared in order to make it suitable for a convolutional neural network. Recurrent neural network and convolutional neural network architectures have shown to perform well in different topics in natural language processing such as text classification, sentiment analysis, machine translation, summarization and others. When handling data with text, the input type to these models varies from characters based tokens, word based tokens up to sentence based tokens. Which one to use depends on the problem and they have different advantages and disadvantages. For example character based models have the advantage of flexibility with unknown tokens and they don’t suffer from out-of-vocabulary problem since all tokens can be build with characters. While this can be a significant advantage, character level based input becomes infeasible in recurrent neural networks for large input since the time series for the model would be very large and also inefficient, not mentioning extremely long training times. Character based convolutional neural networks have shown to perform well in several topics in natural language processing.

Character based inputs will be prepared and evaluated in convolutional neural network
and in order to find the model best suited for the given problem in this thesis.

Before proceeding with preparing character based and word based inputs, preprocessing of the root dataset shown in table 3 will occur, starting with removing all the GPU call stacks since they don’t provide any useful information, bringing down the total number of samples to 5.2M from 5.8M. Next, using a list of irrelevant function names we manage to shrink number of samples to 4.8M from 5.2M. The format of the dataset has yet not be changed from the format shown in table 3, however there are now a total of 4.8M samples instead of 5.8M. The next step consists of preparing the data in a format compatible with character based convolutional neural network.

### 3.3 Character quantization

#### 3.3.1 1D-Convolution

For the character based CNN, we are going to create an input which consist of a sequence of encoded characters, inspired by [81]. For each report, we will append all the rows from the call stack containing function names and create a matrix with the shape (#feature, #feature length) (See figure 9). The number of features will be determined by our character dictionary which will consist of legal characters in C++ and C# which sum up to 72 characters. The character § is used to denote unknown characters in the call stacks (which occur sometimes). Lastly, the character ø will be used between each function name when appending the rows to facilitate learning process of the model. Function names which are more recent in the call stack are more likely to be relevant for finding the reason for the fault than function names in the beginning of the call stack. Therefore, the appending of the function names for the input feature will be done in reversed order which will facilitate learning for the model. In [81], they use a feature length of 1014 and any character exceeding that length is ignored. They also quantize unknown characters as all-zero vectors. For our problem and dataset we will use a maximum feature length of 2077 from maximum 50 rows of function names, since this is the length of 99% of the call stacks in our dataset, both in terms of maximum rows and maximum feature length when appending all the 50 function name rows. Call stacks longer than 2077 characters will be truncated in the beginning as opposed to [81] where they instead truncate in the end. Truncating in the beginning is motivated by how recent rows in the call stack are often more relevant for finding the cause of the different faults.
3.3.2 2D-Convolution

Further experiments will include convolutions with 2D-convolutions. The input for 2D-convolution will be created in a form of a 3D matrix. The matrix will have 50 rows of function names with each row having a maximum length of 128. The depth of the matrix will be the width of the embedding layer which will be 73. 73 is the size of our vocabulary. Each call stack input will thus have the shape (50,128,73) (see figure 8), this will make the input compatible with 2D-convolutions.

3.4 Data Augmentation

Data augmentation is a useful technique to improve the performance of artificial neural networks. It is widely used to create additional artificial data together with the original dataset. This leads to an increased amount of samples in the dataset and hence, a better chance of minimising generalisation error for deep learning models. Data augmentation is highly applicable in different input types such as images and sound in image/speech recognition. It is also applicable when the input is text in other areas, but text augmentation is not as straightforward as handling data with images or sounds. For example, manipulating and augmenting images can be done by:

- Flipping images horizontally and vertically.
- Scaling images by zooming in or zooming out.
- Rotating images. For example by rotating 90 degrees.
4 Experiments

4.1 Dataset

The distribution of the 4.7M dataset $CS_{4.7M}$ is very challenging in that a vast majority of the cluster_ids only have one sample, i.e. only one input per unique class. This makes it hard for any neural network to approximate the underlying function for this dataset since it lacks variance per unique class. A subset of the dataset will be used in the experiments to investigate the method of measuring distance in the Latent Space. The subset will contain cluster_ids which have at least two or more samples per cluster_id. See figure 10, Note that the majority of cluster_ids have only two unique samples per class.
Figure 10: Distribution of $CS_{1.5K}$. Y axis is number of unique samples per class. X axis is each unique cluster_id. Note that the majority of cluster_ids only have two unique samples per class.

We will now refer the subset as dataset $CS_{1.5K}$ since it contains a total of 1500 reports with 410 unique cluster_ids. This is the dataset that will be used in the experiments below. $CS_{1.5K}$ will be split in 33% testing dataset and the remaining 67% as training dataset and will also be shuffled. To guarantee the same split for all the models, a fixed seed number will be used which will ensure the same input for all different models. This is also useful to get reproducible results.

### 4.2 Model architectures

There are no strict guidelines when designing model architectures given just the problem description. One common approach which we have taken in this project, is to start with prior experience of networks with similar problem. One source of good knowledge and experience for these problems are research papers which can provide a good starting point for designing model architectures. We have tried different models architectures and made modifications to make the models more suitable for our specific problem.
In addition to this we have experimented each model architecture with many different hyper-parameters and settings such as number of hidden layers, number of nodes, different kind of activation functions etc. Designing model architectures by trial-and-error is a time-consuming task and often requires sufficient hardware to be able to run good amount of experiments in a reasonable time.

Model1X is partially inspired from [81] with some modification. Model3X, 7X, 7XY, 8X and 8XY are partially inspired from [14] with modifications.

### 4.2.1 Custom Layer StackConv

![StackConv Layer](image)

Figure 11: StackConv Layer.

The StackConv layer (see 11) is a common block in all our models and is the first layer. It consists of an embedding layer and a convolutional layer. The input to the embedding layer is call stacks with shape (50,128).
4.2.2 Model1X, Model3X, Model4X, Model5X and Model6X

Model1X consists of 22 layers. Starting with StackConv layer followed by convolutional, dropout and MaxPooling layers in two blocks. This is followed by 4 blocks of convolutional with Dropout. The next layer is a flatten layer followed by a Dropout layer. The last 4 layers consist of two dense layers with a Dropout between and finally the Softmax layer. The convolutional layers have 32, 64 and 128 filters. The first two convolution layers have kernel size of 7 and the last 4 convolutional layers have kernel size of 3. All Dropout layers have a drop rate of 0.2 except for the last two which have a drop rate of 0.5. ReLu activation was used after each convolutional. Model1X is partially inspired from [81] with modifications. This architecture was intended for a larger dataset, hence the deep architecture.

Model3X is the deepest model with 45 layers (see figure 13). The filter size of the
convolutional layers is 16, 16, 16, 32, 32, 64, 64 with all of them having kernel size of 3. The core block of this model is convolutional layer followed by BatchNormalization layer followed by Dropout layer. This block is repeated 8 times with some of them having MaxPooling layer as well. The model has two dense layers with a Dropout layer between with a Softmax layer as the last layer. Dropout layers have a drop rate of 0.2 except for the Dropout layer between dense layers which have 0.5. This architecture is partially inspired by [14] with modifications.

Model4X consists of 20 layers, it is similar to Model3X and has the same layers in the first 11 layers. Also like Model3X the last layers consist of a flatten, Dense, dropout, Dense and Softmax layer. All Dropout layers have a drop rate of 0.2 except the last dropout between dense layers which has a drop rate of 0.5. No MaxPooling layer was used in this model and like the previous model, LeakyReLu activation was used.

The most shallow architecture of those in figure 13 is Model5X with a total of 13 layers. Starting with a StackConv layer, it has two blocks of convolutional, dropout and MaxPooling layer. This is followed by convolutional, dropout and flatten layer. This model only has one dense layer before the Softmax layer. All Dropout layers have a drop rate of 0.3 and ReLu activation was used in this model.

Model6X is built with 28 layers with 6 convolutional layers, 6 BatchNormalization layers, 9 dropout layers, 2 MaxPooling layers, 1 flatten layer, 2 dense layers and one final Softmax layer. All Dropout layers have a drop rate of 0.3 and ReLu activation was used.

In all of these models above, different number of units was experimented in the dense layers, ranging from 16,32,64,128 to 256. Batch Size in the interval of [8, 16, 32, 64, 128, 160, 256] was used.
4.2.3 Model7X, Model7XY, Model8X and Model8XY

Figure 13: Model7X, Model7XY, Model8X and Model8XY architecture
Model7X, Model7XY, Model8X and Model8XY are deep architectures with a total of 42 layers. The only difference between Model7 and Model 8 is that Model7 uses GlobalMaxPooling layer and no Flatten layer in the end. The difference between X and XY models is that the XY model has a much higher number of filters in each convolutional layer. These models all use kernel size of 3 in all convolutional layers and dropout with a rate of 0.2 with the exception for the last Dropout layer which has a rate of 0.5. These deep architectures were intended for larger dataset only.

Model9-Model14 convolutional layers used filter size in the interval [8,16,32,64,96]. Kernel sizes of [2,3,4] were experimented with. The dense layers had hidden units ranging in the interval of [8, 16, 32, 64, 128, 256]. The models are briefly described below.
4.2.4 Model9

Model9 consists of 8 layers with 1 StackConv, 1 BatchNormalization, 1 MaxPooling, 1 Dropout, 1 Flatten, 1 Dense and a final Softmax layer. Drop rate of 0.2 was used in this model.

Figure 14: Model9 architecture.
4 Experiments

4.2.5 Model10

The slightly deeper model10 consists of 12 layers and 13 layers with Batch Normalization parameter set to true. It has 1 StackConv, 1 (2) BatchNormalization, 2 MaxPooling,
4 Experiments

2 Dropout, 1 Convolutional, 1 Flatten, 1 Dense and a final Softmax layer. Dropout layer used a drop rate of 0.2. LeakyReLU activation was used.
4.2.6 Model11

Figure 16: Model11 architecture.
Model11 consist of 16 layers and 18 layers with Batch Normalization parameter set to true. It is similar to model10 but with one extra block of convolutional, MaxPooling and Dropout layer. Drop rate of 0.2 on all Dropout layers.

4.2.7 Model12

Figure 17: Model12 architecture.
Model12 have 20 layers and 25 layers with Batch Normalization parameter set to true. It is similar to model11 expect that model12 have 2 dense layers with one Dropout layer between with a drop rate of 0.5 instead of only 1 dense layer.

4.2.8 Model13

![Model13 Diagram]

Figure 18: Model13 architecture.

Model13 consists of 12 layers and 14 layers with Batch Normalization parameter set to true. It has 1 StackConv, 1 (2) BatchNormalization, 2 AveragePooling, 1 Convolutional, 1 Flatten, 2 Dense and a final Softmax layer. No Dropout layer was used in this model. LeakyReLu activation was used.
4.2.9 Model14

Model14 has 19 layers and 23 layers with Batch Normalization parameter set to true. It has 1 StackConv, 1 (3) BatchNormalization, 3 MaxPooling, 4 Dropout, 2 Convolutional, 1 Flatten, 2 Dense and a final Softmax layer. Drop rate of 0.2 on all Dropout layers except the last one which had 0.5. LeakyRelu activation was used in this model.

Figure 19: Model14 architecture.
4.3 Hardware

The following hardware was used in these experiments:

- GPU: 2 GeForce GTX TITAN X, 3072 CUDA Cores, 12 GB GDDR5 Memory
- RAM: 64 GB
- CPU: Intel Core i7-5960X, 8 cores, 16 threads, 3.00 GHz

4.4 Software

In this section, we will briefly describe the software used for experiments.

For fetching data from the database, the Python module Pyodbc was used. Pyodbc is a Python package which can be used to connect to ODBC databases.

The Python library NumPy [74] was used to preprocess the data from the database and it was also used for saving and loading datasets. NumPy, also unofficially called Numerical Python [59] is one of the most common and fundamental package in Python for scientific computing.

Pandas [54] is another fundamental and powerful Python package used for data analysis [52] [53]. It provides efficient data structures that are easy to use with high performance. This makes it also suitable for data manipulation and visualisation. Pandas were used in this thesis to efficiently collect and visualise the result from the experiments, facilitating the work to merge and organise all the results from the experiments.

Scikit-Learn [60] is a machine learning library in Python which provides support of various unsupervised and supervised algorithms and methods commonly used in machine learning. In this thesis, it was used to split up the dataset into training and testing set in an appropriate fashion.

4.4.1 Keras

Keras is a high abstraction framework in Python used for creating neural networks, created by François Chollet [12]. It has the advantage of being user-friendly and hence allow for fast experimentation. Keras can run by using the backend from TensorFlow[1] [2], CNTK [69], or Theano [7][5][3]. We have only used TensorFlow backend in our experiments.
5 Results

The result of top five experiments per model can be seen in table 6, 7 and 8 in terms of testing accuracy, considering only testing accuracy with 0.3 or higher. See table 4 for abbreviations. In the first column, we have each specific model followed by an abbreviation that denotes specific settings in the architecture of the model. The second column specifies Batch Size used which denotes the number of samples per gradient update. The third column 'Layers' specify the total number of layers which exist in the model architecture used in the experiment for each row. The fourth column LS_Dimension specify the dimension used in the fully connected dense layers in each model architecture (except the last softmax dense layer which have 410 units which represent the classes). Model parameters specify the number of trainable parameters for each specific experiment. Lower value means less computation needed to train, lower risk of overfitting etc.

The column 'ES Epoch' stands for Early Stopping which indicate the epoch which ES got triggered. In all our experiment we used Early Stopping with the patience of 5, meaning that the model would stop training if no improvement were made after 5 epochs in terms of validation loss.

The last two columns denote the performance of each experiment in terms of comparing the similarity between call stacks. The similarity metric between call stacks was achieved by feeding call stacks from the training set into the model. Each unique input created a representation in the Latent Space (dense layers) which we treated as clusters. Next, we fed in call stack from the testing set and observed where in the Latent Space it ended. Lastly, we calculated the Euclidean distance from each input from the testing set to all the clusters created from the training set in the Latent Space. For each call stack sample in the testing set, we created a list of five clusters which had the shortest Euclidean distance. If the correct cluster existed in any of these top 5 candidates, then we labeled the prediction as correct. This is how we defined SC_Testing accuracy and C_Testing accuracy. The testing accuracy is the accuracy of the model predicting the true bucket for a given input.
5 Results

Table 4: List of abbreviation used in figures below

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>d0.X</td>
<td>dX stands for Dropout with a rate of X. For example d0.2 stands for Dropout with a rate of 0.2, meaning that 20% of input units drops randomly.</td>
</tr>
<tr>
<td>c0.X</td>
<td>cX stands for convolutional filter which is the dimensionality of the output space. (i.e. the number of output filters in the convolutional). X is an integer for that dimensionality.</td>
</tr>
<tr>
<td>ckX</td>
<td>ck stands for convolutional kernel which specifying the height and width of the 2D convolutional window. X is an integer specifying the height and width.</td>
</tr>
<tr>
<td>bn</td>
<td>bn stands for batch normalization. Models with bn in their name have a batch normalization layer after each convolutional layer.</td>
</tr>
<tr>
<td>ES</td>
<td>ES stands for Early Stopping which indicate the epoch which ES got triggered.</td>
</tr>
<tr>
<td>LS_Dimension</td>
<td>Latent Space Dimension is the dimension (i.e. Number of hidden units) in the last fully connected dense layer/layers. See 4.2.</td>
</tr>
<tr>
<td>SC, Testing Accuracy</td>
<td>SC stands for Supercluster which contain a larger group of Clusters which in turn contains reports. Clusters get manually merged into one bigger Supercluster by software engineers when the Clusters are considered similar or duplicates. This accuracy does take Jira classes into consideration.</td>
</tr>
<tr>
<td>C, Testing Accuracy</td>
<td>C stands for Cluster which contains reports related to that Cluster/Bucket. This accuracy doesn’t take Jira classes into consideration.</td>
</tr>
</tbody>
</table>

Table 5: Top 5 from model1X-model4X with Adam optimizer

<table>
<thead>
<tr>
<th>Model</th>
<th>Batch Size</th>
<th>ES Epoch</th>
<th>Latent Space Dimension</th>
<th>Model Parameters</th>
<th>Total Training Time(s)</th>
<th>Training Accuracy</th>
<th>Testing Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model1X_d0.1</td>
<td>64</td>
<td>22</td>
<td>25</td>
<td>128</td>
<td>1606059</td>
<td>104,0204</td>
<td>0.6175</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>22</td>
<td>82</td>
<td>64</td>
<td>993963</td>
<td>165,4367</td>
<td>0.5452</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>22</td>
<td>53</td>
<td>256</td>
<td>2854827</td>
<td>100,2105</td>
<td>0.6145</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>22</td>
<td>37</td>
<td>256</td>
<td>2854827</td>
<td>79,6665</td>
<td>0.6084</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>22</td>
<td>56</td>
<td>256</td>
<td>2854827</td>
<td>94,1973</td>
<td>0.6888</td>
</tr>
<tr>
<td>Model2X_d0.2</td>
<td>192</td>
<td>31</td>
<td>88</td>
<td>1024</td>
<td>2135371</td>
<td>87,4850</td>
<td>0.7249</td>
</tr>
<tr>
<td></td>
<td>192</td>
<td>31</td>
<td>121</td>
<td>512</td>
<td>875851</td>
<td>113,2500</td>
<td>0.7711</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>31</td>
<td>92</td>
<td>1024</td>
<td>2135371</td>
<td>89,8750</td>
<td>0.6868</td>
</tr>
<tr>
<td></td>
<td>224</td>
<td>31</td>
<td>85</td>
<td>1024</td>
<td>2135371</td>
<td>84,5470</td>
<td>0.6978</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>31</td>
<td>65</td>
<td>1024</td>
<td>2135371</td>
<td>73,1870</td>
<td>0.7380</td>
</tr>
<tr>
<td>Model3X_d0.1</td>
<td>256</td>
<td>45</td>
<td>53</td>
<td>128</td>
<td>773243</td>
<td>87,5731</td>
<td>0.8112</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>45</td>
<td>26</td>
<td>256</td>
<td>1448571</td>
<td>62,1621</td>
<td>0.8112</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>45</td>
<td>36</td>
<td>128</td>
<td>773243</td>
<td>110,0134</td>
<td>0.8062</td>
</tr>
<tr>
<td></td>
<td>96</td>
<td>45</td>
<td>33</td>
<td>256</td>
<td>1448571</td>
<td>66,7325</td>
<td>0.8865</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>45</td>
<td>51</td>
<td>256</td>
<td>1448571</td>
<td>87,8011</td>
<td>0.9498</td>
</tr>
<tr>
<td>Model3X_d0.2</td>
<td>96</td>
<td>45</td>
<td>56</td>
<td>512</td>
<td>2897531</td>
<td>90,0620</td>
<td>0.9890</td>
</tr>
<tr>
<td></td>
<td>224</td>
<td>45</td>
<td>45</td>
<td>1024</td>
<td>6188667</td>
<td>74,9220</td>
<td>0.9618</td>
</tr>
<tr>
<td></td>
<td>384</td>
<td>45</td>
<td>70</td>
<td>256</td>
<td>1448571</td>
<td>103,0470</td>
<td>0.9418</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>45</td>
<td>69</td>
<td>128</td>
<td>773243</td>
<td>108,2500</td>
<td>0.9578</td>
</tr>
<tr>
<td></td>
<td>96</td>
<td>45</td>
<td>43</td>
<td>1024</td>
<td>6188667</td>
<td>75,2190</td>
<td>0.9679</td>
</tr>
<tr>
<td>Model4X_d0.2</td>
<td>96</td>
<td>45</td>
<td>56</td>
<td>512</td>
<td>2897531</td>
<td>90,0620</td>
<td>0.9890</td>
</tr>
<tr>
<td></td>
<td>224</td>
<td>45</td>
<td>45</td>
<td>1024</td>
<td>6188667</td>
<td>74,9220</td>
<td>0.9618</td>
</tr>
<tr>
<td></td>
<td>384</td>
<td>45</td>
<td>70</td>
<td>256</td>
<td>1448571</td>
<td>103,0470</td>
<td>0.9418</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>45</td>
<td>69</td>
<td>128</td>
<td>773243</td>
<td>108,2500</td>
<td>0.9578</td>
</tr>
<tr>
<td></td>
<td>96</td>
<td>45</td>
<td>43</td>
<td>1024</td>
<td>6188667</td>
<td>75,2190</td>
<td>0.9679</td>
</tr>
</tbody>
</table>

Table 5: Top 5 from model1X-model4X with Adam optimizer
### Results

Table 6: Top 5 from model5X-model8XY with Adam optimizer

<table>
<thead>
<tr>
<th>Model</th>
<th>Batch Size</th>
<th>Epochs</th>
<th>Latent Space Dimension</th>
<th>Model Parameters</th>
<th>Total Training Time(s)</th>
<th>Training Accuracy</th>
<th>Testing Accuracy</th>
<th>SC_Testing Accuracy</th>
<th>C_Testing Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model5X_d0.1</td>
<td>32</td>
<td>13</td>
<td>256</td>
<td>1149099</td>
<td>66,4255</td>
<td>0.9679</td>
<td>0.5793</td>
<td>0.7561</td>
<td>0.7541</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>13</td>
<td>32</td>
<td>426251</td>
<td>88,5921</td>
<td>0.9498</td>
<td>0.5813</td>
<td>0.7703</td>
<td>0.7683</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>13</td>
<td>64</td>
<td>529515</td>
<td>133,2281</td>
<td>0.9598</td>
<td>0.5854</td>
<td>0.7703</td>
<td>0.7561</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>13</td>
<td>32</td>
<td>426251</td>
<td>85,2664</td>
<td>0.9709</td>
<td>0.5874</td>
<td>0.7846</td>
<td>0.7846</td>
</tr>
<tr>
<td>Model5X_d0.2</td>
<td>32</td>
<td>13</td>
<td>16</td>
<td>374619</td>
<td>113,5235</td>
<td>0.9468</td>
<td>0.5874</td>
<td>0.7703</td>
<td>0.7642</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>13</td>
<td>22</td>
<td>374619</td>
<td>115,9850</td>
<td>0.9829</td>
<td>0.6179</td>
<td>0.7805</td>
<td>0.7805</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>13</td>
<td>64</td>
<td>529515</td>
<td>134,7190</td>
<td>0.9859</td>
<td>0.6301</td>
<td>0.7663</td>
<td>0.7642</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>13</td>
<td>58</td>
<td>529515</td>
<td>125,0150</td>
<td>0.9799</td>
<td>0.6301</td>
<td>0.7866</td>
<td>0.7805</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>13</td>
<td>16</td>
<td>374619</td>
<td>104,6250</td>
<td>0.9789</td>
<td>0.6484</td>
<td>0.7825</td>
<td>0.7764</td>
</tr>
<tr>
<td>Model5X_d0.3</td>
<td>16</td>
<td>13</td>
<td>63</td>
<td>529515</td>
<td>112,8750</td>
<td>0.9809</td>
<td>0.6159</td>
<td>0.8008</td>
<td>0.7927</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>13</td>
<td>70</td>
<td>426251</td>
<td>112,1880</td>
<td>0.9849</td>
<td>0.6179</td>
<td>0.7846</td>
<td>0.7825</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>13</td>
<td>89</td>
<td>529515</td>
<td>187,0630</td>
<td>0.9749</td>
<td>0.6220</td>
<td>0.7785</td>
<td>0.7622</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>13</td>
<td>108</td>
<td>16</td>
<td>374619</td>
<td>166,2350</td>
<td>0.9769</td>
<td>0.6260</td>
<td>0.7947</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>13</td>
<td>91</td>
<td>32</td>
<td>426251</td>
<td>190,3280</td>
<td>0.9849</td>
<td>0.6362</td>
<td>0.7907</td>
</tr>
<tr>
<td>Model6X_d0.2</td>
<td>160</td>
<td>28</td>
<td>55</td>
<td>256</td>
<td>3756939</td>
<td>77,0310</td>
<td>0.3785</td>
<td>0.3028</td>
<td>0.5650</td>
</tr>
<tr>
<td>Model6X_d0.3</td>
<td>128</td>
<td>28</td>
<td>63</td>
<td>256</td>
<td>3756939</td>
<td>142,7660</td>
<td>0.7821</td>
<td>0.4837</td>
<td>0.6687</td>
</tr>
<tr>
<td>Model8X_d0.2</td>
<td>64</td>
<td>42</td>
<td>96</td>
<td>1024</td>
<td>39950827</td>
<td>238,6570</td>
<td>0.5181</td>
<td>0.3150</td>
<td>0.7195</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>42</td>
<td>123</td>
<td>512</td>
<td>20603371</td>
<td>404,7660</td>
<td>0.5080</td>
<td>0.3394</td>
<td>0.6565</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>42</td>
<td>127</td>
<td>512</td>
<td>20603371</td>
<td>320,0310</td>
<td>0.6004</td>
<td>0.4370</td>
<td>0.6911</td>
</tr>
<tr>
<td>Model8XY_d0.3</td>
<td>32</td>
<td>42</td>
<td>66</td>
<td>1024</td>
<td>10952075</td>
<td>121,4530</td>
<td>0.7269</td>
<td>0.4512</td>
<td>0.7561</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>42</td>
<td>92</td>
<td>1024</td>
<td>10952075</td>
<td>136,3120</td>
<td>0.8012</td>
<td>0.4776</td>
<td>0.8049</td>
</tr>
<tr>
<td></td>
<td>192</td>
<td>42</td>
<td>86</td>
<td>1024</td>
<td>10952075</td>
<td>115,7190</td>
<td>0.6627</td>
<td>0.4858</td>
<td>0.7459</td>
</tr>
<tr>
<td></td>
<td>96</td>
<td>42</td>
<td>93</td>
<td>512</td>
<td>5367179</td>
<td>124,5620</td>
<td>0.5291</td>
<td>0.4939</td>
<td>0.7378</td>
</tr>
<tr>
<td></td>
<td>224</td>
<td>42</td>
<td>88</td>
<td>1024</td>
<td>10952075</td>
<td>117,3900</td>
<td>0.7420</td>
<td>0.5467</td>
<td>0.7500</td>
</tr>
</tbody>
</table>
### Table 7: Top 5 from model9-model14 with Adam optimizer

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Model9.cf64_ck2</td>
<td>16</td>
<td>8</td>
<td>34</td>
<td>128</td>
<td>12970411</td>
<td>73.3390</td>
<td>0.9970</td>
<td>0.5691</td>
<td>0.6829</td>
<td>0.6768</td>
</tr>
<tr>
<td>Model9.cf32_ck3_bn</td>
<td>16</td>
<td>8</td>
<td>20</td>
<td>256</td>
<td>13020523</td>
<td>62.4070</td>
<td>0.9920</td>
<td>0.5711</td>
<td>0.6626</td>
<td>0.6484</td>
</tr>
<tr>
<td>Model9.cf64_ck3</td>
<td>16</td>
<td>8</td>
<td>17</td>
<td>64</td>
<td>6497579</td>
<td>36.3910</td>
<td>0.9960</td>
<td>0.5711</td>
<td>0.6992</td>
<td>0.6850</td>
</tr>
<tr>
<td>Model9.cf96_ck3</td>
<td>32</td>
<td>8</td>
<td>30</td>
<td>64</td>
<td>9730347</td>
<td>57.2650</td>
<td>0.9960</td>
<td>0.5732</td>
<td>0.6992</td>
<td>0.6931</td>
</tr>
<tr>
<td>Model9.cf64_ck3_bn</td>
<td>64</td>
<td>8</td>
<td>32</td>
<td>64</td>
<td>6497579</td>
<td>43.5000</td>
<td>1.0000</td>
<td>0.5772</td>
<td>0.7053</td>
<td>0.6931</td>
</tr>
<tr>
<td>Model10.cf32_ck4</td>
<td>32</td>
<td>12</td>
<td>28</td>
<td>128</td>
<td>3148107</td>
<td>40.5780</td>
<td>0.9980</td>
<td>0.6301</td>
<td>0.7683</td>
<td>0.7561</td>
</tr>
<tr>
<td>Model10.cf96_ck4</td>
<td>16</td>
<td>12</td>
<td>37</td>
<td>128</td>
<td>9524235</td>
<td>114.5160</td>
<td>1.0000</td>
<td>0.6301</td>
<td>0.7886</td>
<td>0.7805</td>
</tr>
<tr>
<td>Model10.cf64_ck4</td>
<td>32</td>
<td>12</td>
<td>27</td>
<td>128</td>
<td>6303403</td>
<td>53.0160</td>
<td>1.0000</td>
<td>0.6341</td>
<td>0.7866</td>
<td>0.7703</td>
</tr>
<tr>
<td>Model10.cf96_ck4</td>
<td>32</td>
<td>12</td>
<td>70</td>
<td>256</td>
<td>18719115</td>
<td>189.5000</td>
<td>1.0000</td>
<td>0.6382</td>
<td>0.7846</td>
<td>0.7764</td>
</tr>
<tr>
<td>Model10.cf64_ck4</td>
<td>16</td>
<td>12</td>
<td>61</td>
<td>256</td>
<td>18719115</td>
<td>213.3440</td>
<td>1.0000</td>
<td>0.6423</td>
<td>0.7967</td>
<td>0.7927</td>
</tr>
<tr>
<td>Model11.cf64_ck4</td>
<td>16</td>
<td>16</td>
<td>33</td>
<td>256</td>
<td>6683947</td>
<td>102.5209</td>
<td>0.9980</td>
<td>0.6707</td>
<td>0.8293</td>
<td>0.8171</td>
</tr>
<tr>
<td>Model11.cf96_ck4</td>
<td>16</td>
<td>16</td>
<td>22</td>
<td>64</td>
<td>3747531</td>
<td>97.1295</td>
<td>0.9950</td>
<td>0.6707</td>
<td>0.8333</td>
<td>0.8293</td>
</tr>
<tr>
<td>Model11.cf96_ck4</td>
<td>32</td>
<td>16</td>
<td>62</td>
<td>128</td>
<td>5985675</td>
<td>228.1580</td>
<td>1.0000</td>
<td>0.6728</td>
<td>0.8354</td>
<td>0.8313</td>
</tr>
<tr>
<td>Model11.cf96_ck4</td>
<td>32</td>
<td>16</td>
<td>68</td>
<td>256</td>
<td>10461963</td>
<td>255.0851</td>
<td>1.0000</td>
<td>0.6789</td>
<td>0.8415</td>
<td>0.8293</td>
</tr>
<tr>
<td>Model11.cf96_ck4</td>
<td>16</td>
<td>16</td>
<td>38</td>
<td>128</td>
<td>5985675</td>
<td>166.0024</td>
<td>1.0000</td>
<td>0.6992</td>
<td>0.8537</td>
<td>0.8496</td>
</tr>
<tr>
<td>Model12.cf96_ck4</td>
<td>32</td>
<td>20</td>
<td>37</td>
<td>64</td>
<td>10829675</td>
<td>274.1654</td>
<td>0.9548</td>
<td>0.6423</td>
<td>0.7927</td>
<td>0.7744</td>
</tr>
<tr>
<td>Model12.cf64_ck4</td>
<td>32</td>
<td>20</td>
<td>38</td>
<td>128</td>
<td>13004907</td>
<td>166.4383</td>
<td>0.9950</td>
<td>0.6443</td>
<td>0.7764</td>
<td>0.7602</td>
</tr>
<tr>
<td>Model12.cf32_ck4</td>
<td>16</td>
<td>20</td>
<td>28</td>
<td>64</td>
<td>3273579</td>
<td>81.3493</td>
<td>0.9438</td>
<td>0.6443</td>
<td>0.7785</td>
<td>0.7703</td>
</tr>
<tr>
<td>Model12.cf96_ck4</td>
<td>32</td>
<td>20</td>
<td>45</td>
<td>128</td>
<td>20010603</td>
<td>340.5613</td>
<td>0.9980</td>
<td>0.6463</td>
<td>0.7886</td>
<td>0.7764</td>
</tr>
<tr>
<td>Model12.cf96_ck4</td>
<td>96</td>
<td>20</td>
<td>69</td>
<td>64</td>
<td>10829675</td>
<td>472.2234</td>
<td>0.9890</td>
<td>0.6504</td>
<td>0.7927</td>
<td>0.7846</td>
</tr>
<tr>
<td>Model13.cf96_ck2_bn</td>
<td>16</td>
<td>14</td>
<td>40</td>
<td>128</td>
<td>9306827</td>
<td>119.8605</td>
<td>1.0000</td>
<td>0.6504</td>
<td>0.7561</td>
<td>0.7398</td>
</tr>
<tr>
<td>Model13.cf32_ck3_bn</td>
<td>16</td>
<td>14</td>
<td>39</td>
<td>256</td>
<td>6298539</td>
<td>79.4492</td>
<td>1.0000</td>
<td>0.6504</td>
<td>0.7866</td>
<td>0.7642</td>
</tr>
<tr>
<td>Model13.cf96_ck3</td>
<td>160</td>
<td>12</td>
<td>71</td>
<td>128</td>
<td>9404715</td>
<td>142.4678</td>
<td>1.0000</td>
<td>0.6504</td>
<td>0.7866</td>
<td>0.7846</td>
</tr>
<tr>
<td>Model13.cf96_ck4_bn</td>
<td>16</td>
<td>14</td>
<td>63</td>
<td>256</td>
<td>18786699</td>
<td>249.5949</td>
<td>1.0000</td>
<td>0.6524</td>
<td>0.7886</td>
<td>0.7703</td>
</tr>
<tr>
<td>Model13.cf96_ck4</td>
<td>32</td>
<td>14</td>
<td>38</td>
<td>128</td>
<td>9542027</td>
<td>117.0436</td>
<td>1.0000</td>
<td>0.6565</td>
<td>0.7744</td>
<td>0.7602</td>
</tr>
<tr>
<td>Model14.cf96_ck4</td>
<td>64</td>
<td>19</td>
<td>43</td>
<td>128</td>
<td>9329067</td>
<td>206.1745</td>
<td>0.9950</td>
<td>0.6504</td>
<td>0.7785</td>
<td>0.7663</td>
</tr>
<tr>
<td>Model14.cf96_ck2</td>
<td>16</td>
<td>19</td>
<td>30</td>
<td>128</td>
<td>8651499</td>
<td>133.0246</td>
<td>0.9980</td>
<td>0.6504</td>
<td>0.7825</td>
<td>0.7744</td>
</tr>
<tr>
<td>Model14.cf96_ck2</td>
<td>32</td>
<td>19</td>
<td>25</td>
<td>128</td>
<td>8651499</td>
<td>98.6168</td>
<td>0.9940</td>
<td>0.6504</td>
<td>0.7724</td>
<td>0.7642</td>
</tr>
<tr>
<td>Model14.cf96_ck3</td>
<td>32</td>
<td>19</td>
<td>43</td>
<td>128</td>
<td>8934987</td>
<td>162.5814</td>
<td>1.0000</td>
<td>0.6565</td>
<td>0.7744</td>
<td>0.7663</td>
</tr>
<tr>
<td>Model14.cf64_ck4</td>
<td>32</td>
<td>19</td>
<td>39</td>
<td>256</td>
<td>11505131</td>
<td>133.0553</td>
<td>0.9990</td>
<td>0.6585</td>
<td>0.7927</td>
<td>0.7764</td>
</tr>
</tbody>
</table>
We have experimented with several architectures of convolutional neural networks with many different configurations in order to investigate the method of measuring similarity between call stacks in the latent space. We have experimented with architectures consisting of 8 layers and deep architectures consisting up to 45 layers. The number of model parameters varies from 60k to 40M and Adam and Nadam optimizers have been experimented with. The results were slightly better with Nadam optimizer (See table 8 and table 6).

In terms of predicting the right bucket among 410 classes, we achieved a testing accuracy of 70%. This is a number that can be improved greatly by having more data and also better higher quality of data. Model 11 achieved the best result in terms of testing accuracy.
Figure 20: (a) training accuracy, (b) training loss. Model9-Model14 (except model10) with adam optimizer. Epoch on x-axis.

accuracy, SC_testing accuracy and C_testing accuracy.

The results shows that measuring call stack similarity in the latent space is possible. For each call stack input our model can recommend the top five buckets that have shortest Euclidean distance to the input. Out of these five recommendations, the model predicts the true bucket id with 86.18% testing accuracy.

In the older models, Model5X with dropout of 0.2 achieved the best testing accuracy. Model3X with dropout of 0.1 achieved best performance in terms of SC_testing accuracy and C_testing accuracy. (See table 6). The models with much deeper architectures were designed for larger dataset and not the CS1.5K dataset.

The result of measuring distance with Mahalanobis distance was not as useful as we thought. We believe that this was due to the characteristics of CS1.5K dataset having mostly two samples per call stack instance. Hence, no sufficient groupings were formed in latent space because of the lack of variance in the input. The consequence of the model forming points instead of groups in the latent space might be one of the reasons why euclidean distance was more applicable than Mahalanobis distance. We believe
Figure 21: (a) testing accuracy, (b) testing loss. Model9-Model14 (expect model10) with adam optimizer. Epoch on x-axis.

that Mahalanobis distance would be more useful if each call stack instance had more samples, leading to reasonable groupings in the latent space. Despite this, the Mahalanobis distance can be useful when determining when a call stack should be assigned to a new bucket by using the distance with a reasonable threshold. For example, if there is a new input to the model and the distance is longer than x from any other known clusters, then it is likely that that input should be assigned to a new bucket (see figure 24). This would be very useful for the existing system and would lead to reduced duplicate reports and also reduced number of incorrectly grouped reports. This needs to be investigated in future experiments.
Figure 22: (a) training accuracy, (b) training loss. Model9-Model14 with nadam optimizer. Epoch on x-axis.

Figure 24: Illustration of possible concept for determining when a new bucket should be created using a suitable threshold.

We also implemented a custom loss when we trained the models. We used KL-divergence
7 Future work

The characteristics of the dataset heavily constrains the applicability of our method. For better results a bigger dataset with more instances per bucket would be preferable. Having a bigger dataset with more variance among buckets would allow us to experiment with more complex architectures, achieve better generalisation and also most likely increase the applicability of measuring distance using Mahalanobis distance.

It is important to investigate the model in production to see if it is suitable for detecting...
outliers for new input. Detecting emerging new classes is an important step in order to reducing duplicate reports. Another important factor is having the right accuracy to prevent incorrectly grouped reports. For example having the right threshold when using Mahalanobis distance for determining whether an input belongs to a cluster or should be assigned a new bucket.

The database which the call stacks were fetched from contained other information from each report which could have been used for further strengthening the input for better results. This is something worth investigating in the future. However, finding other relevant features for the input is a time-consuming task and also requires domain-expertise which is not always available.

Future works also include experimenting with other methods than neural networks, for example decision trees[66]. Another possible approach is to use a combination of different methods.

References


and nonneuronal cells make the human brain an isometrically scaled-up primate brain,” *Journal of Comparative Neurology*, vol. 513, no. 5, pp. 532–541, 2009.


References


