***INM427: A Comparative Study in Bank Customers Churn Predictions using ANNs and CNNs***

Ibrahim.n’diaye@city.ac.uk

**Abstract:** This study provides a comparison between artificial neural networks (ANNs) and convolutional neural networks (CNNs) in predicting bank customer churn. Both ANNs and CNNs were trained on the dataset and evaluated based on different performance metrics. We start by pre-processing the data by removing columns which have a low correlation and then converting highly correlative data to tensors. CNNs demonstrated proficiency in identifying crucial features that contribute to churn. The study recommends that ANNs are better suited for structured data, while CNNs can be useful in identifying key factors affecting churn. These findings can help banks enhance their customer retention strategies and reduce customer churn rates. To work on this further, we could look at RNNs and SVMs too.

**Introduction**

Banks are concerned about customer churn as it can result in loss of revenue and market share. Hence, it is crucial for banks to predict potential churners and take proactive measures to retain them. Customer churn is a critical problem faced by the banking industry. Customer churn can result in revenue losses, increased marketing costs, and reduced profitability. Therefore, predicting customer churn accurately is vital for banks to maintain their customer base and increase profitability. Predictive modelling techniques, including artificial neural networks (ANNs) and convolutional neural networks (CNNs), have been extensively used across industries, including banking, for predicting customer churn [1].

The aim of this study is to compare the effectiveness of ANNs and CNNs in predicting bank customer churn. The study employs a dataset of bank customers that contains demographic and transactional information labelled with churn or non-churn status. Both ANNs and CNNs are trained on this dataset, and their performance is evaluated by constructing a confusion matrix and an ROC curve, then using F1 score, accuracy, precision and recall.

While ANNs are commonly used for structured data and have shown promising results in various predictive modelling tasks, CNNs are typically used for image classification but have also been used for other types of data, including structured data. By comparing ANNs and CNNs, this study provides insights into which technique is more effective for predicting customer churn in the banking industry. These findings can help banks enhance their customer retention strategies, reduce customer churn rates, and increase their market share.

This paper will first discuss the related work on customer churn prediction in the banking industry and provide an overview of ANNs and CNNs. Next, we will describe the dataset and methodology used in this study, followed by a presentation of the results of the comparative analysis. Finally, we will discuss the implications of the findings and suggest directions for future research.

**Hypothesis Statement**

We are going to predict due to the nature of our data being more structured and less visual, that ANNs will outperform CNNs and have a lower accuracy/ precision/ recall loss.

**Convolutional Neural Networks (CNNs)**

Convolutional Neural Networks (CNNs) have shown great potential in predicting customer churn due to their ability to learn features automatically from large datasets. The study by [2] provides evidence that CNNs can be effective in predicting bank customer churn. Banks can leverage CNNs to build accurate customer churn prediction models that can help reduce churn rates, increase revenue, and improve profitability. Additionally, the study highlights the importance of collecting and using transaction history data in customer churn prediction models.

Chart

Description automatically generated

*Figure 1: Architecture of a CNN*

CNNs contain four different types of layers, as seen above. The purpose of an input layer in a convolutional neural network is to receive input data to pass through the other layers. Input layer normally consists of a tensor. We then work through convolutional layers which have a set of filters which are applied to the input vector (in this case we are not using an image), performing a convolutional operation and highlight the import data points as part of the tensor. We then have a pooling layer, which reduces the size of the feature map created from the convolutional layer by down sampling to make the network more efficient. Max Pooling layers which are what we use in our CNN models take the maximum value in each region and discard the rest. After pooling, the remaining modified input vectors are then passed through fully connected layers which perform a linear transformation on them before passing them through the output layer, which consists of an activation function that helps the network understand non-linearity of the data and understand complex relationships better.

The advantages are effective for finding patterns and features in large, complex datasets which are useful for predicting fraud, they can learn complex patterns and relevant features from raw data ignoring the need for feature engineering. On the other hand, they are computationally expensive and require significant computing power to train and run, which can cost monetary issues. Furthermore, there are ethical concerns with using CNNs for bank data due to their potential bias causing discrimination to certain groups or individuals. Nevertheless, they can handle large datasets effectively which caters for faster processing and analysis. However, they are not as transparent or interpretable as traditional statistical models.

**Neural Networks (ANNs)**

Diagram, schematic

Description automatically generated

The traditional Neural Network is the baseline of deep learning, modelling input data and transmitting it through a series of interconnected processing nodes applying linear functions before producing the output. The idea of this is to mimic how the brain of a human processes information and combined with computational efficiency, this can be a very powerful framework.

The ANN has an activation function, input layer, hidden layers and an output layer. The input layer consists of data in the form of a vector which is passed onto the hidden layer. During this training process, the ANN adjusts the weight of the connections between neurons to minimise the difference between the predicted output and the actual output, using an optimisation algorithm and a loss function (in this case Stochastic Gradient Descent and binary cross-entropy loss. After being passed through the hidden layers, we reach the output layer where the final prediction is produced.

They are capable for identifying complex and non-linear relationships between variables, which is useful for analysing data. They are more flexible than CNNs and can generalise unseen data more easily. They are less likely to overfit than CNNs and can interpret data slightly better. On the other hand, they are less computationally efficient than ANNs, especially with larger data and are less able to process spatial relationships between images or time-series data.

**Dataset Analysis + Pre-Processing**

The dataset used in the study is Bank churn datasets typically contain historical transactional data of customers, along with a label indicating whether the customer has churned or not. The data is usually collected over a specific period, such as a few months or a year, and includes information on customer demographics, account details, and transactional behaviour. The datasets contain 10000rows x 14 cols

Timeline

Description automatically generated

After analysing the correlation matrix of the variables in our dataset, we exclude the variables below 0.05 correlation, and we found that we just used “Age”, “Balance”, “IsActiveMember” and “Gender” along with the dependent variable “Exited” in our final dataset.

**Exploratory Data Analysis (EDA)**

A picture containing chart

Description automatically generated

From these two Kernel Density plots, the top one analysing Age and the bottom analysing Balance (two continuous variables), we can see that it is more likely that younger people are more likely to leave the bank. People who are older are likely to be more loyal. It is common for people to exit the bank at 35.

Chart, bar chart

Description automatically generatedFrom these countplots, we can see that more males are likely to stay, and more females are likely to leave the bank; this bank is more male-dominant. We can also conclude that active members are have tendency to leave more than non-active members implying the bank needs to improve its services and target its active members and younger age groups more.

Summary Statistics Table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Variable | Mean | Standard Deviation | Minimum | Maximum |
| Age | 38.92 | 10.49 | 18.0 | 92 |
| Balance | 76485.89 | 62397.41 | 0 | 250898.10 |
| IsActiveMember | 0.52 | 0.50 | 0 | 1 |
| Gender | 0.55 | 0.50 | 0 | 1 |
| Exited | 0.20 | 0.40 | 0 | 1 |

**Methods**

We elaborate on the methodology utilised during training, validation and testing steps for CNN and ANN models. Additionally, an in-depth description of architecture and hyper-parameter implementation will be provided.

**Methodology**

After data pre-processing, involving feature selection creating a new data frame, then applying the “min-max” scaler to the numerical variables and the label encoder to the categorical variables, we start preparing our data for the train/validation/test split. We use the min-max scaler to preserve the original distribution of the dataset and relationship between features and original values while making it more amenable for deep learning algorithms. Since the categorical variables are binary (either 0 or 1) and do not have an order, we prefer the use of a label encoder over a one-hot encoder.

Moving on, we apply the train-test-split function twice on the data with the ratio given ({train: 0.70, validation: 0.15, test: 0.15}). The proposed methodology involves partitioning the original dataset into training and testing sets, with 15% of the data reserved for testing purposes. The remaining 85% is used for both model selection and algorithm comparison between Convolutional Neural Networks (CNNs) and Artificial Neural Networks (ANNs) with 15% being used as a validation set for hyper-parameter tuning.

The next step involves changing these 6 datasets into numpy arrays and then tensors. It is important to change these to tensors because then they can be used with skorch and pytorch implementations to run the model.

For skorch implementations, we leave the data as tensors and run the models on validation data, but for pytorch, we convert the tensors into tensor datasets and then data loaders to be inputted into the training loop along with the model.

We now move on to constructing the architecture for our models:

**Architecture and Hyper-Parameters used for the CNN**

For the CNN, the key hyper-parameters we use are learning rate, batch size, number of epochs, momentum and activation functions. Since we are predicting categorical data (whether or not they have exited), we will use Sigmoid activation function as it is most appropriate. Another alternative would be Softmax however this is used preferably for multi-class classification. Lastly, tanh is a viable option.

|  |  |
| --- | --- |
| **Names** | **Values** |
| Learning Rate | 0.01, 0.1, 0.5 |
| Batch Size | 500, 1000, 2000 |
| Number of Epochs | 10, 25, 50 |
| Momentum | 0.8, 2.0, 4.0 |
| Activation Functions | Sigmoid, Tanh, Softmax |

We utilise quite a complex CNN architecture. We start of with a simple input layer, before moving onto the convolutional layer section of the network. This is responsible for performing the convolution operations on the input data, learning local patterns and temporal dependencies present. These convolutions will extract

Convolutional Layers

This consists of 3 layers with the following specifications:

* The first layer has 3 input channels, 32 output channels and a kernel size of 3
* The second layer has 32 input channels, 64 output channels and a kernel size of 3
* The final layer has 64 input channels, 128 output channels and a kernel size of 3

We keep the kernel size the same, but the standard is 32-64-128 for the channels in this section. Each convolutional layer has a kernel size of 2 and a stride of 2. Max-Pooling layers divide the input feature map into non-overlapping regions and then take the maximum value within each region, in doing so they make CNNs more computationally efficient and reduce overfitting.

We the move on to the fully connected layers, these consist of 2 hidden layers and an output layer; the first hidden layer consists of 128 units and a dropout probability of 0.2, whereas the second one has a dropout probability of 0.4 with 64 units. Finally, the output layer has 2 units (for 0 and 1).

**Architecture and Parameters used for the ANN**

Similar for the ANN, we will look at the learning rate, batch size, number of epochs, momentum and activation functions. We can also modify other hyper-parameters, such as the dropout however we keep this at 0.5 for now. Further analysis will be done using a grid-search on the number of hidden layers, to identify which is the best optimal hyper-parameter combination for our final model.

|  |  |
| --- | --- |
| **Names** | **Values** |
| Learning Rate | 0.01, 0.1, 0.5 |
| Batch Size | 500, 1000, 2000 |
| Number of Epochs | 10, 25, 50 |
| Momentum | 0.8, 2.0, 4.0 |
| Activation Functions | Sigmoid, Tanh, SoftMax |
| Number of Hidden Layers | [500, 250], [250, 125], [120, 60] |

Our ANN features a custom number of hidden layers, a dropout value of 0.5 an input layer, hidden layers and output layer. This is the probability that the randomly selected neurons are temporarily removed from the network during training to prevent overfitting, meaning the remaining neurons must learn without assistance of dropped out neurons. We modify the number of hidden layers as a hyper-parameter which are a series of nodes conducting matrix multiplications and linear transformations along with non-linear activation functions. Finally, the output layer gives the final prediction.

The binary cross-entropy loss function and the SGD optimizer both work in the ANN by affecting the weights and biases during training. They introduce non-linearity into the model to provide feedback on how to adjust weights and biases to minimise the loss. The sigmoid function works by introducing non-linearity into the model enabling it to learn complex decision boundaries and improve its accuracy on non-linearly separable data.

**Post Model Creation**

Moving on, we train our models on our training data (training tensors for Skorch, and training data-loaders for PyTorch) before moving on to validating our models. Skorch enables us to use the Sci-Kit Learn grid-search on Pytorch models, enabling us to find the best hyper-parameters more easily, before finally running these models on our test data giving the final accuracy and loss of the optimised models.

We present our results for this Binary Classification problem in a series of ROC curves and confusion matrices.

**Results, Findings & Evaluation**

Confusion Matrices

Confusion Matrix ANN

|  |  |  |
| --- | --- | --- |
|  | True | False |
| True | 34 (True Positive) | 8 (False Positive) |
| False | 13 (False Negative) | 67 (True Negative) |

Confusion Matrix CNN

|  |  |  |
| --- | --- | --- |
|  | True | False |
| True | 54 (True Positive) | 14 (False Positive) |
| False | 18 (False Negative) | 57 (True Negative) |

From our final confusion matrices, we can see that ANN has a higher accuracy of (34+67)/ (34+8 + 13 + 67) = 0.8289 or 82.9% compared to CNNs which have an accuracy of (54 + 57)/ (54 + 14+ 1 + 57) = 0.78 or 78%. In terms of specificity, we have that ANNs had 89.3% compared to 80.28%. Overall we can conclude that in terms of F1 score, Specificity, Precision (80.95% compared to 79.41%). We can conclude that ANNs outperform CNNs once hyper-parameter tuning has been completed.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Learning Rate | Number of  Hidden  Layers | Number of Epochs | Activation Function | Momentum | Batch Size |
| ANN | 0.1 | [500, 250] | 25 | Sigmoid() | 2.0 | 500 |
| CNN | 0.1 | [500,250] | 10 | Sigmoid() | 2.0 | 500 |

This is the final architecture for both architectures. We can see that most of the activation functions, momentum and batch sizes are the same, with a smaller batch size resulting in the least loss. This is due to the size of the data, only having 10,000 values in total.

**Model Selection**

Chart, line chart

Description automatically generated  
Running both our final models on our test dataset over 8 epochs, we can see that the ANN loss is slightly lower than the CNN loss throughout, further backing up our initial hypothesis. This is perhaps due to the CNN being more likely to overfit the data due to its complexity, we could combat this by introducing a higher dropout value and more Max-Pooling layers to minimise overfitting.

**Conclusions**

We can confirm our hypothesis statement and say that ANNs outperform CNNs overall.

**Lessons Learnt**

The main error we were struggling with included modifying the tensors so the pytorch implementation worked with the Data-Loaders. The tensors within this data need to be in a certain format [batch\_size, num\_features, num\_channels]. We encountered multiple errors initially with train, test and validation tensor sizes having to be updated and reshaped. Also, for this specific dataset, we can say that ANNs outperformed CNNs, however with careful hyper-parameter tuning, CNNs can perform ANNs. This highlights the significance of hyper-parameter tuning.

**Future Work**

We could experiment with different optimizers and try the adam optimizer too. Further experimentation can be done with modifying the dropout probability in these neural networks. Other networks we can look at for this sort of problem involve RNNs, SVM (Support Vector Machines) , MLP (Multi-Layer Propogation), RBMs. Further study could also be done into these models to evaluate the effectiveness on this dataset. Lastly, a comparison with machine learning models could also be done.

**6. References**

[1] Zhong & Li - Proceedings of the 2019 3rd International Conference on Innovation in Artificial Intelligence – 2019

[2] Zhang, Y., Li, X., Li, Y., & Li, X. (2021). Predicting Bank Customer Churn with Convolutional Neural Networks. Journal of Risk and Financial Management, 14(2), 72. <https://doi.org/10.3390/jrfm14020072>

[3] Jiang, Y., Zhang, L., & Song, X. (2020). Bank Customer Churn Prediction Model Based on Artificial Neural Network. Journal of Physics: Conference Series, 1623(1), 012095.

[4] Kuo, Y. F., Wu, C. M., & Deng, W. J. (2017). A neural network approach for customer churn prediction in subscription services. Expert Systems with Applications, 76, 125-134.

[6] J. Bergstra and Y. Bengio, “Random search for hyper-parameter optimization,” Journal of Machine Learning Research, vol. 13, pp. 281–305, 2012.

[7] J. L. Ba, J. R. Kiros, and G. E. Hinton, “Layer normalization,” arXiv preprint arXiv:1607.06450, 2016.