Sales Time Series Forecasting using a Hybrid CNN-LSTM Model

Zachary Russell
za443739@ucf.edu
University of Central Florida
Orlando, Florida, USA

ABSTRACT

Time series forecasting of grocery sales is an important aspect of ensuring brick-and-mortar grocery stores avoid unwanted food waste, poor profit margins, and wasted chances to capitalize on promotional opportunities. Time series forecasting has been used for other applications like financial, weather, stock price, and energy forecasting. This paper proposes utilizing a hybrid CNN-LSTM to conduct sales time series forecasting on data provided from a Kaggle competition. With an optimized architecture and hyperparameters, the model was able to produce great results, scoring a mean absolute error of 58.26 and a root mean squared error of 257.26. This model outperformed other types of long short-term memory models and gated recurrent units. The use of convolution layers in conjunction with a long short-term memory layer has shown that it does improve the ability to perform time series forecasting of sales values.

CCS CONCEPTS

• **Computing methodologies** → **Model development and analysis**; **Neural networks**.

KEYWORDS

CNN, LSTM, Neural Network, Recurrent Neural Network, Time-Series, Forecasting, Sales, Predictions

ACM Reference Format:
Zachary Russell. 2024. Sales Time Series Forecasting using a Hybrid CNN-LSTM Model. In ,. ACM, New York, NY, USA, 8 pages.

1 INTRODUCTION

In the field of statistics and modeling, time series forecasting is the practice of using previously collected data over a historical period of time and using it to predict future values. The most common uses found for it include strategic decision-making for businesses, weather forecasting, finance forecasting and more. Different machine learning methods have become adept at performing time series forecasting for a multitude of use-cases. Methods using Random Forest (RF), Support Vector Regression (SVR), Gated Recurrent Units (GRU), Artificial Neural Networks (ANN), and Long Short-Term Memory (LSTM) models have all been used for time series forecasting[11].

Long Short-Term Memory models are type of recurrent neural network proposed by Schmidhuber et al. in 1997[4]. This type of model has a well-researched reputation of performing accurately across time series forecasting applications. LSTMs have been shown to effectively handle large amounts of data, achieve success in sequential data applications, and outperform other RNN models because of its ability to bypass the gradient descent problem other RNNs encounter[10]. Convolutional Neural Networks (CNN) are typically used in image processing and image classification applications. CNNs excel at image processing because of their ability to perform feature extraction on images, however it has been shown that CNNs can also perform useful feature extraction on normalized data [1]. Between a CNNs ability to process data quickly and extract relevant features for regression, and an LSTMs ability identify patterns over large periods of time, a hybrid model combining the qualities of both needed to have its potential uses researched. This paper aims to use a hybrid CNN-LSTM model to perform accurate time series forecasting of grocery sales using data given from a Kaggle competition.

2 RELATED WORK

In recent years there has been other research into the use of hybrid CNN-LSTM models for time series forecasting. No paper found has been used for predicting sales values for a business. The existing papers consists of using this type of model for financial time series forecasting, stock price time series forecasting, energy forecasting, and others.

In October 2020, Lu et. al [8] proposed a CNN-LSTM model for forecasting stock prices. Stock prices can be influenced by a myriad of external factors that need to be considered when attempting to predict the future stock price. Due to the large volume of features that are needed to make accurate predictions, the researchers proposed this type of model in order to achieve more accurate predictions than other proposed models. The results of the paper showed the CNN-LSTM model outperformed other models including a Multi-Layer Perceptron, a CNN-RNN, a basic RNN, a simple CNN, and a simple LSTM.

A paper utilizing a CNN-LSTM model to predict residential energy consumption was published in September of 2019 [6]. Due to the special nature of energy consumption, the researchers needed a model that could extract both spatial and temporal features from the input data to accurately predict housing energy consumption. These predictions are used to ensure that power plants adjust their output to account for the potential energy consumption. The paper proved that the model outperformed previous models of varying architecture used for the same application.

Other research used a CNN-LSTM hybrid model to predict the price fluctuation of the gold standard. [7] proposed a model of two convolution layers, followed by a max pooling layer that feeds into the LSTM. The paper's results concluded that the model did perform better with the addition of the convolution layers than other time series forecasting models that just utilized LSTM layers.

Each of these related works highlight the performance improvement achieved when combining LSTM models with convolutional layers. This paper will aim to achieve the same goals as these related works but instead for sales predictions of brick-and-mortar grocery stores.

3 APPROACH

3.1 Proposed Model

The model architecture utilized a series of convolution layers in conjunction with 1 LSTM layer and a time-distributed dense layer. Figure 2 visualizes the architecture of the model in detail.

The first convolution layer in this model contains 64 filters with a kernel size of 3 and a ReLU activation function. The ReLU activation function is defined in Equation 1. The ReLU activation takes the output from the convolutions and returns 0 for any negative values, and the original value for non-negative values.

$$f(x) = \max(0, x) \tag{1}$$

This convolution layer is followed by a batch normalization layer. The batch normalization layer applies a transformation on the data that maintains the mean output near 0 and the output standard deviation near 1. The batch normalization layers operate differently during training and predicting. During training, the layer normalizes its outputs by using the standard deviation of the current input batch. The layer performs the computation described in Equation 2 for each channel being normalized [5].

$$y = \gamma \frac{(\text{batch} - \mu(\text{batch}))}{\sqrt{\text{var}(\text{batch}) + \epsilon}} + \beta \tag{2}$$

Where ϵ is a small constant to avoid dividing by zero, γ is a learned scaling factor, and β is a learned offset factor. During inference, this layer instead normalizes it's outputs using a moving average of the standard deviation and mean of the batch seen while training. That changes the computation seen in Equation 2 to the computation defined by Equation 3.

$$y = \gamma \frac{(\text{batch} - \text{moving\_mean})}{\sqrt{\text{moving\_var} + \epsilon}} + \beta \tag{3}$$

Following the batch normalization layer is a single dropout layer. This dropout layer changes 20% of the values passed through it to zeros. This layer is vital to the training process because it helps the model understand how each feature from the input data affects the results of the output data, thus allowing the model to form better relationships between features and their impact on sale values. It is important to note that the dropout layer is only active during training, during predicting the dropout layer is not used and is instead skipped.

The output of the previous dropout layer feeds directly into another convolution layer. This layer contains 128 filters, a kernel size of 9 and a ReLU activation function. A greater number of layers is used here to extract as many useful features possible before being fed into the LSTM layer. This larger convolution layer is also followed by its own batch normalization layer and another dropout layer both with parameters equal to the ones in the aforementioned paragraphs.

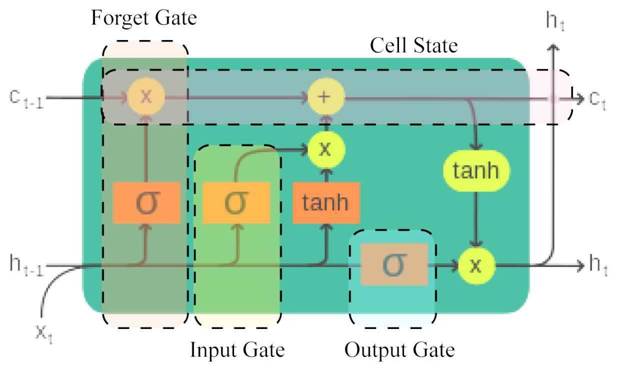

Figure 1: Architecture of LSTM Memory Unit

The LSTM layer takes the input data from the previous layers and feeds it through its recurrent neural network architecture. LSTMs are able to retain values that are key in producing accurate outputs, and forgetting values that are not important to accurate outputs. A LSTM unit is composed of a cell, input gate, output gate, and forget gate as outlined in Figure 1[2].

The LSTM layer was configured with a dimensionality output space of 50. The LSTM was also configured to use dropout for the linear transformation of the inputs and the linear transformation of the current state. A drop out value of 20% was used for both of these transformations. The tanh function in the LSTM memory cell is defined by Equation 4 and was the activation function chosen for the LSTM layer.

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{4}$$

The output of the LSTM layers is connected to a time-distributed dense layer. The dense layer outputs a dimensionality space of 33 to match the shape of the data on the Y axis.

3.2 Data

The data provided by the Kaggle competition consisted of 6 different datasets:

- train.csv
- test.csv
- oil.csv
- stores.csv
- holidays_events.csv
- transactions.csv

The datasets span a data range of January 1, 2013 to August 31, 2017. The only exception being the split of the training and testing datasets. The split ends the training set at August 15, 2017 and the testing set begins at August 16, 2017. The *train.csv* dataset contained a series store numbers, sales values of 33 categories of products, and the number of items in that category that were on promotion that respective date. The *test.csv* dataset only contained the store numbers and the number of items on sale in specific product categories on the respective date. The *oil.csv* dataset contained the daily oil

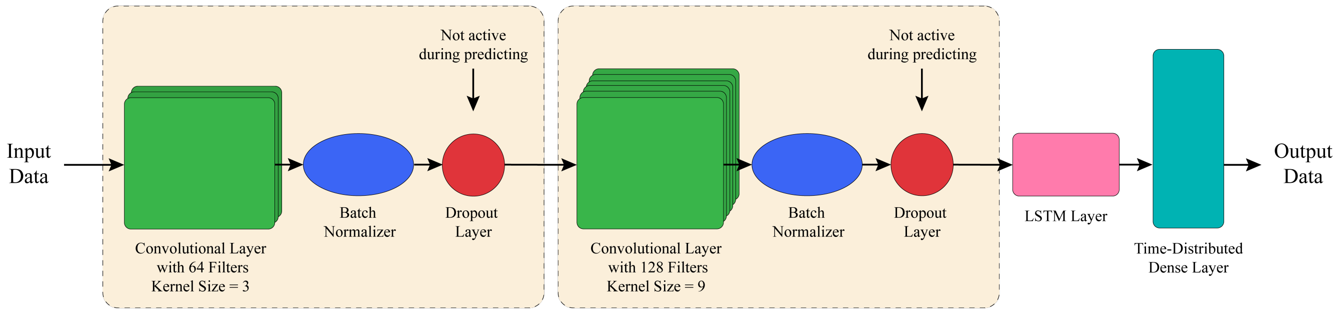
Input Data
Convolutional Layer with 64 Filters Kernel Size = 3
Batch Normalizer
Not active during predicting
Dropout Layer
Convolutional Layer with 128 Filters Kernel Size = 9
Batch Normalizer
Not active during predicting
Dropout Layer
LSTM Layer
Time-Distributed Dense Layer
Output Data

Figure 2: CNN-LSTM model architecture used for grocery sales time series forecasting

price from January 1, 2013 to August 31, 2017. *Stores.csv* contained regional information for each store including an ID number, their city and state location, the type of store, and what cluster of stores each individual store belonged to. The final dataset used was the *holidays_events.csv*, which contained a list of dates pertaining to important holidays both national and regional in the country. If the holiday was regional, the holiday's city was included in the dataset. The transactions dataset was not used for training as there was no inclusion of transaction numbers with the test dataset, therefore it would not be possible to match the input shapes for both training and testing.

3.3 Data Pre-Processing

All of the datasets previously discussed needed to be combined into a cohesive set of input data that could be passed to the proposed model. All pre-processing steps described hereafter were applied to both the training and the testing datasets.

The training and testing set was originally structured with each store taking up 33 rows for 1 date. This is because there are 33 different categories of product that need to be predicted. This resulted in 54 stores taking 33 rows each for 1 date in the training and testing dataset; resulting in 1684 rows of data for a single date and a total dataset length of 3,000,888 rows for training and 28,512 for testing. The datasets were restructured such that for each store in the dataset, their 33 product categories were grouped into a single row for each date. This reduced the training dataset length down to 90,936 rows and the testing dataset down to 864 rows. This now meant that every 54 rows in the dataset equaled 1 day or 1 time-step.

Before the oil dataset could be combined with our training and testing datasets, it needed missing dates and oil prices filled in to avoid NaN values in the input data. All dates missing in the date range were inserted into their respective places. All missing oil prices were filled in using a rolling average with a window size of 15, so the price of oil 7 days before and 7 days after the missing date were used to fill in the missing value. The oil dataset was then merged with the training set aligned to the *date* feature so oil prices were duplicated across all 54 rows in each time-step. Holiday events were extracted from the *holiday_events* dataset. From these dates a new feature was engineered and added to both the train and test dataset called *has_holiday*. This binary feature was used to mark whether the date in the dataset was a holiday or not. Additional to the *has_holiday* feature, 4 more features were engineered to signify the anticipation leading up to a holiday. A marker for three weeks before, two weeks before, one week before, and 3 days before were all created as binary features. If a date met the criteria for any of these 4 attributes, a 1 was put in the respective column, if not, a zero.

A combination of label encoding and one-hot encoding was used for various attributes in the datasets given. The city and state location of the stores as well as the store type were label encoded into numerical representations. The ID value given to each store known as *store_nbr* was one-hot encoded. This was done to avoid this feature from being interpreted by the model of having significance on the outcome of the predictions. The date column in both the training and testing dataset were split into three separate columns called month, date and year for easier interpretation by the model. An additional column was feature engineered that signified the day of the week as well.

With all datasets combined into a single data frame, the data frame could now be split into its X and Y axes and scaled. The Y features included only the 33 product categories that were to be predicted, every other feature remained apart of the X features. Once separated, each axes was scaled using a Min-Max scaling. The formula for min-max scaling is defined by Equation 5.

$$m = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \tag{5}$$

where m denotes the new value, x is the original value, $x_{\min}$ is the minimum value in the entire column, and $x_{\max}$ is the maximum value in the entire column.

Min-max scaling needed to be performed on the data before it was given to the CNN-LSTM model. This type of scaling allows the data to be transformed to a common scale. This transformation also allowed the data to then be reshaped for time series forecasting. Before reshaping, the training X data held a shape of `(90936, 101)` and the training Y data held a shape of `(90936, 33)`. The CNN-LSTM model expects a 3-dimensional shape of `[samples, time-steps, features]`. A total of 54 stores' 33 product categories are being predicted every time step. So, the *timesteps* value in the shape is 54 and the *sample* value must be reduced down by the number of time-steps. After reshaping, the training data X has a shape of `[1684, 54, 101]` and the training Y data holds a shape of `[1684, 54, 33]`. With this shape structure, the CNN-LSTM model will be able to evaluate all stores every time-step and output the 33 product category sales predictions per store.

3.4 Model and Hyperparameter Fine-Tuning

Complicated external factors all contribute to the sales of different item categories and total sales at brick-and-mortar stores. Due to this, the CNN-LSTM model this paper proposes required adjustments and heavy fine-tuning during training iterations. To fine-tune the model, both the Mean Absolute Error (MAE) metric and the Root Mean Squared Error (RMSE) was used to measure the model's predictions against the actual sales. The mean absolute error metric is defined in Equation 6 and the root mean squared error metric is defined in Equation 7. After each model architecture and parameter adjustment, the MAE and RMSE were calculated and compared with previous configurations. The goal of this fine-tuning step is to get the MAE and RMSE as small as possible.

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{6}$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n} (y_i - \hat{y}_i)^2} \tag{7}$$

3.4.1 Model Architecture Adjustments. The first model parameters adjusted during experimentation was the kernel sizes of both the 64 filter convolution layer and the 128 filter convolution layer. Kernel sizes of 3, 9, and 12 were tested on each layer in different combinations. It was found that a kernel size of 3 for the 64 filter convolution layer and a kernel size of 9 for the 128 filter convolution layer achieved the lowest MAE and RMSE score. For the LSTM layer, only the dimensionality space was adjusted during experimentation. Dimensionality space values of 25, 50, 75, and 100 were all tested on the LSTM layer with the optimal convolutional layer settings. The best value found for the LSTM dimensionality space was 50.

3.4.2 Hyperparameter Fine-Tuning. Hyperparameters for this model included the type of optimizer used during training, the learning rate used, and the batch size used. A range of optimizers were tested including Adagrad, RMSprop, Nadam, and Adam. All optimizers were tested using the final model architecture discussed in the previous section and displayed in Figure 2. Out of the 4 optimizers, Nadam resulted in the best score metrics. Nadam uses the same approach as the Adam optimizer, however it incorporates Nesterov Momentum to improve the optimization process [3].

Using the Nadam optimizer, learning rate values of 0.01 to 0.0001 were used to train the model. It is important to note that an early stopping technique was used during training, this allowed the model to train until it reached the loss and validation loss minima without continuing too far past it. Out of the values tested, 0.001 was found to be the most efficient learning rate and produced the lowest score metrics.

Batch sizes of 32, 64, and 128 were tested during model training. Each batch size had significant changes on the outcomes of the model performance. It was decided a batch size of 32 was chosen as the best batch size for the model to examine the input data. Details of all model tests using various model architectures and

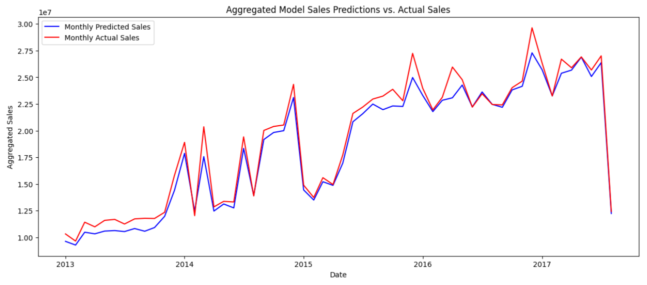

Figure 3: Model Sales Predictions vs. Actual Sales aggregated monthly

hyperparameter fine-tuning are outlined in Table 1. Where C1F is convolution filter 1, CF2 is convolution filter 2, CAF is convolution activation function, Ker is kernel size, and LSTM AF is LSTM activation function.

4 EXPERIMENTATION

To pre-process the datasets outlined in Section 3.2 and perform the model tuning, training, and testing as outlined in Section 3.4, the experimentation was done using a Google Colaboratory instance. The Google Colaboratory instance was a Python 3 Google Compute Engine backend with 12.7GB of RAM and 225.8GB of disk space.

4.1 Model Performance

After training with the optimized parameters discovered in Section 3.4, the model generated predictions on the training validation data and was evaluated against the actual validation sales values. The model achieved an MAE score of 58.26 and an RMSE score of 257.26. The validation dataset has numbers of sales that averaged around approximately 50,000 with a peak close to 140,000 sales. In context with the amount of actual sales, the predicted's metric scores are measurably adequate and are well within the ability to accurately predict the number of sales for these grocery stores. Figure 3 shows the models predicted sales plotted with the actual sales from the training dataset aggregated monthly for legibility. A more detailed graph of the predicted sales and actual sales on a daily basis can be seen in Figure 4.

4.2 Model Comparison

This paper's CNN-LSTM model was compared against three other models that are common for time series forecasting problems. Each of these models were rated using the same metrics to allow for a fair comparison with the proposed model. Each of the models architecture will be briefly described and their metrics will be shared. Each of these models predictions were also charted with the datasets actuals in the Appendix.

4.2.1 Basic LSTM. A baseline LSTM was first used to compare against the research's proposed model. This model was used to see if the convolution layers in the proposed model actually aided in performance. The basic LSTM consisted of a single LSTM layer and a single time-distributed dense layer. The LSTM in this model has a dimensionality output of 75, and the time-distributed dense

Table 1: Architecture and Hyperparameter Fine-Tuning: Sorted in descending order by MAE score

CF1	Ker 1	CAF. 1	CF2	Ker 2	CAF. 2	LSTM Units	LSTM AF.	Optimizer	Learning Rate	Batch Size	MAE	RMSE
64	9	ReLU	128	9	ReLU	75	tanh	Adagrad	0.001	32	412.20	1364.11
64	9	ReLU	128	9	ReLU	75	tanh	RMSprop	0.001	32	94.99	349.73
64	9	ReLU	128	9	ReLU	100	tanh	Nadam	0.001	32	75.84	301.83
64	12	ReLU	128	9	ReLU	75	tanh	Nadam	0.001	32	70.00	288.96
64	9	ReLU	128	9	ReLU	50	tanh	Nadam	0.001	64	70.85	283.13
64	9	ReLU	128	3	ReLU	75	tanh	Nadam	0.001	32	71.34	292.37
64	9	ReLU	128	9	ReLU	50	tanh	Nadam	0.001	128	66.90	277.63
64	3	ReLU	128	3	ReLU	75	tanh	Nadam	0.001	32	67.48	286.25
64	9	ReLU	128	9	ReLU	50	tanh	Nadam	0.01	32	68.77	283.10
64	9	ReLU	128	9	ReLU	25	tanh	Nadam	0.001	32	68.77	283.87
64	9	ReLU	128	9	ReLU	50	tanh	Nadam	0.0001	32	67.55	274.51
64	9	ReLU	128	9	ReLU	75	tanh	Adam	0.001	32	64.73	271.81
64	12	ReLU	128	12	ReLU	75	tanh	Nadam	0.001	32	63.75	267.62
64	3	ReLU	128	12	ReLU	75	tanh	Nadam	0.001	32	63.28	265.03
64	9	ReLU	128	9	ReLU	75	tanh	Nadam	0.001	32	60.72	258.40
64	9	ReLU	128	9	ReLU	50	tanh	Nadam	0.001	32	60.46	266.12
64	3	ReLU	128	9	ReLU	75	tanh	Nadam	0.001	32	59.56	260.46
64	9	ReLU	128	9	ReLU	50	tanh	Nadam	0.001	32	59.53	261.48
64	3	ReLU	128	9	ReLU	50	tanh	Nadam	0.001	32	58.26	257.26

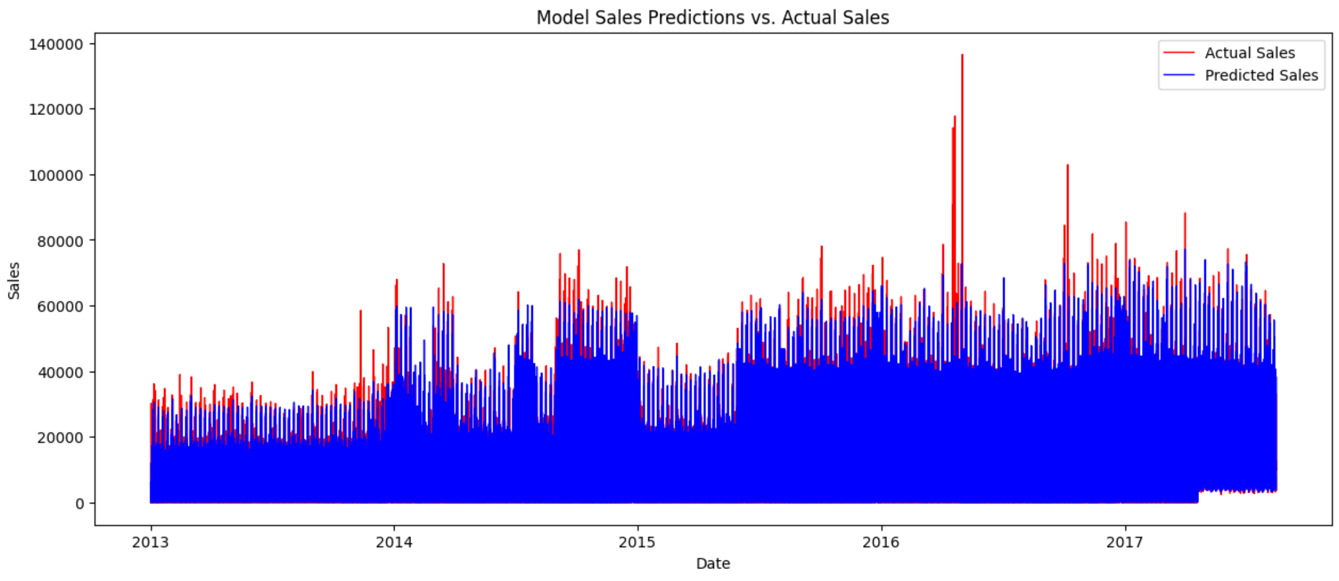

Figure 4: Model Sales Predictions vs. Actual Sales

layer output a dimensionality space of 33. This model achieved an MAE score of 77.85 and an RMSE score of 306.63. This model's score metrics resulted in much larger values than the proposed CNN-LSTM. The predicteds vs actuals charts for this basic LSTM model can be found in Appendix A.

4.2.2 Bidirectional LSTM. A special variant of LSTM was also used to evaluate its effectiveness in predicting sales values for this dataset. Bidirectional LSTMs utilize two LSTM layers that iterate through the data in opposite directions. It has been shown that Bidirectional LSTM models can provide better time series predictions than basic LSTMs and even Autoregressive Integrated Moving Average (ARIMA) models [9]. The architecture included the first LSTM layer to iterate forwards with a dimensionality space of 50, and a second to iterate backwards with the same dimensionality space. The backwards LSTM layer was connected to a time-distributed dense layer that output a dimensionality space of 33. This model achieved an MAE score of 69.10 and an RMSE score of 284.40. The score metrics in this model improved upon the basic LSTM but were still worse than this research's CNN-LSTM model. The predicteds vs actuals charts for this model are in Appendix B.

4.2.3 Gated Recurrent Unit (GRU). A gated recurrent unit is type of recurrent neural network similar to an LSTM in the context that it has a gating mechanism to input or forget certain features, however it lacks a context vector; thus, resulting in fewer parameters when compared to an LSTM. GRU models have been found to be efficient and accurate in previous research, even sometimes outperforming LSTM models [12]. The architecture of this model contained a single GRU layer that directly interpreted the input data, and output a dimensionality space of 50. The GRU layer was connected to a time-distributed dense layer that output the final predictions with a dimensionality space of 33. The GRU model achieved an MAE score of 73.63 and an RMSE of 295.79. This model fell between the performance of the basic LSTM and the bidirectional LSTM and still did not outperform the CNN-LSTM model. The charts for this model are listed under Appendix C.

5 DISCUSSION

There were many unsuccessful approaches to the proposed problem that required heavy modification of the intended model. Originally, the model was supposed to solely be convolution layers and an LSTM layer. However this original model failed to capture relationships between the input data and output predictions. This original model was scoring with RMSE values around 40,000. As portrayed in the final proposed model, adding dropout layers after each convolution layer, as well as adding dropout to the LSTM layer significantly improved the performance of the model.

With the addition of the dropout layers, the model was more accurately predicting actual sales with good accuracy, however the model's charted predictions appeared too smooth. Sales of stores regularly spike and drop in accordance to certain factors, an important one being the occurrence of holidays. The model was failing to see a correlation between a gradual increase in sales leading up to a holiday. At this point, the only data pre-processing done to signify holiday events was the single *has_holiday* binary feature that marked whether a date was a holiday or not. In order to better encapsulate the anticipatory purchasing before a holiday, the 4 attributes outlined in Section 3.3 were created. Figure 5 visualizes how the model fails to capture spikes in sales specifically around the end of each year where there are holidays like Christmas and New Year's Eve. When compared to Figure 3, there is a stark difference between the models performance before and after the addition of these 4 features.

When following the monthly actual sales in Figure 3, there is a noticeable spike in approximately April of 2016 that the CNN-LSTM model fails to capture. This is because on April 16, 2016 the country of the stores that the datasets were created from, Ecuador, was struck by an earthquake. Following this natural disaster, a rally of relief support was made to the country which spiked the sales in these grocery stores across country. This spike can also be seen in Figure 4 where it was the highest recorded sales out of the 4 year time period. This clearly caused an abnormality in the dataset provided. If the data pre-processing was changed to account for this, it is believed the MAE and RMSE values would be even lower than the best values achieved during testing. However if the CNN-LSTM model were utilized in a real-world scenario, it would not be able to predict a natural disaster such as this. Thus, no adjustments were made to data pre-processing so the model could account for the occurrence of this earthquake. For experimentation purposes, the model was re-evaluated without this anomalous period of high sales by removing the data from April 16, 2016 to May 15, 2016. Upon evaluation the CNN-LSTM model did perform better with an MAE value of 55.41 and an RMSE of 221.63.

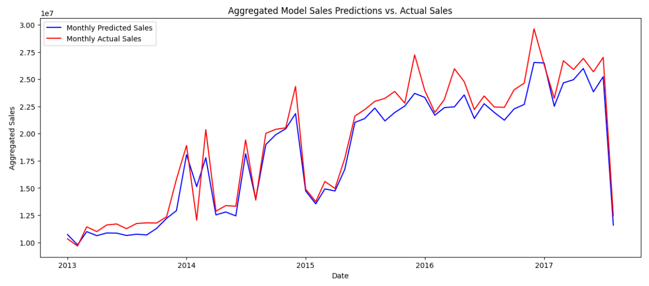

Figure 5: Model Sales Predictions before Holiday Feature Engineering vs. Actual Sales aggregated monthly

6 CONCLUSION

This paper has concluded that a hybrid CNN-LSTM model performs well for time series forecasting, specifically time series forecasting relating to the sales of brick-and-mortar grocery stores. This paper's proposed model outperform a basic LSTM, and bidirectional LSTM, and a GRU model in all performance metrics introduced. With sufficient input data this model could be used in other grocery sales scenarios non-specific to the datasets evaluated in this paper and could benefit businesses greatly. More accurate sales predictions can lead to less food waste, greater profit margins, and better store promotion management.

7 FUTURE WORK

Future iterations on this research could include improving the model farther than it was in this paper. With better hyperparameter fine-tuning this model could perform even better than it previously recorded. A grid search could be used in future applications to perfectly tune the model to its best parameters. A more complex model could also be explored, utilizing more convolution layers or LSTM layers could positively impact the model. Seeing the impact of using a bidirectional LSTM in conjunction with the convolution layers could produce better results, as the bidirectional LSTM performed relatively well in Section 4.2.2 with such basic parameters. The application of CNN-LSTM should not be limited to this specific scenario and should continue to be explored for other time series forecasting applications like ones discussed in the Section 2.

REFERENCES

[1] Yun Bai, Zhiqiang Chen, Jingjing Xie, and Chuan Li. 2016. Daily reservoir inflow forecasting using multiscale deep feature learning with hybrid models. *Journal of hydrology* 532 (2016), 193–206.

[2] Guillaume Chevalier. 2023. LSTM Image. Wikimedia Commons. https://commons.wikimedia.org/wiki/File:LSTM_image.png CC BY-SA 4.0. Available from: https://creativecommons.org/licenses/by-sa/4.0.

[3] Timothy Dozat. 2016. Incorporating nesterov momentum into adam. (2016).

[4] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[5] Keras. 2024. Batch Normalization Layer - Keras Documentation. https://keras.io/api/layers/normalization_layers/batch_normalization/ Accessed: 2024-04-18.

[6] Tae-Young Kim and Sung-Bae Cho. 2019. Predicting residential energy consumption using CNN-LSTM neural networks. *Energy* 182 (2019), 72–81. https://doi.org/10.1016/j.energy.2019.05.230

[7] I. E. Livieris, E. Pintelas, and P. Pintelas. 2020. A CNN–LSTM model for gold price time-series forecasting. *Neural Computing and Applications* 32 (2020), 17351–17360. https://doi.org/10.1007/s00521-020-04867-x

[8] Wenjie Lu, Jiazheng Li, Yifan Li, Aijun Sun, and Jingyang Wang. 2020. A CNN-LSTM-Based Model to Forecast Stock Prices. *Complexity* 2020 (2020), 1–10. https://doi.org/10.1155/2020/6622927 Article ID 6622927.

[9] Sima Siami-Namini, Neda Tavakoli, and Akbar Siami Namin. 2019. The Performance of LSTM and BiLSTM in Forecasting Time Series. In *2019 IEEE International Conference on Big Data (Big Data)*. 3285–3292. https://doi.org/10.1109/BigData47090.2019.9005997

[10] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. *Advances in neural information processing systems* 27 (2014), 9 pages.

[11] Yajiao Tang, Zhenyu Song, Yulin Zhu, Huaiyu Yuan, Maozhang Hou, Junkai Ji, Cheng Tang, and Jianqiang Li. 2022. A survey on machine learning models for financial time series forecasting. *Neurocomputing* 512 (2022), 363–380. https://doi.org/10.1016/j.neucom.2022.09.003

[12] Peter T. Yamak, Li Yujian, and Pius K. Gadosey. 2020. A Comparison between ARIMA, LSTM, and GRU for Time Series Forecasting. In *Proceedings of the 2019 2nd International Conference on Algorithms, Computing and Artificial Intelligence* (Sanya, China) *(ACAI '19)*. Association for Computing Machinery, New York, NY, USA, 49–55. https://doi.org/10.1145/3377713.3377722

A BASIC LSTM MODEL CHARTS

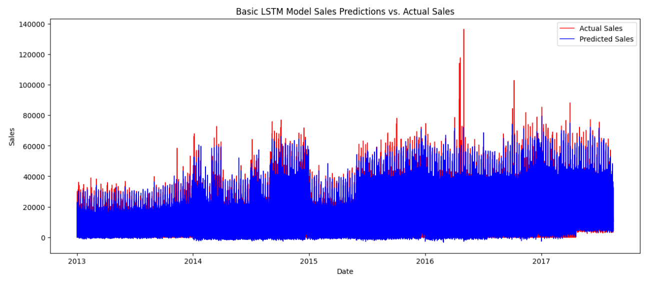

Figure 6: Basic LSTM Model Sales Predictions vs. Actual Sales

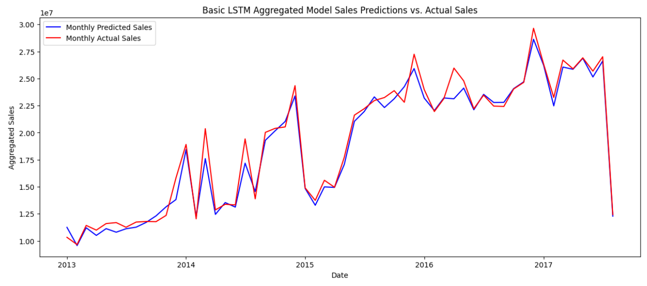

Figure 7: Basic LSTM Model Sales Predictions vs. Actual Sales aggregated monthly

B BIDIRECTIONAL LSTM MODEL CHARTS

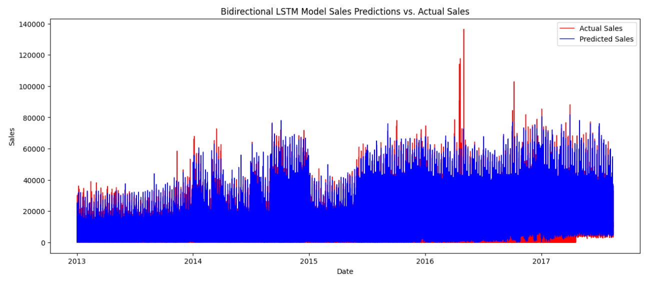

Figure 8: Bidirectional LSTM Model Sales Predictions vs. Actual Sales

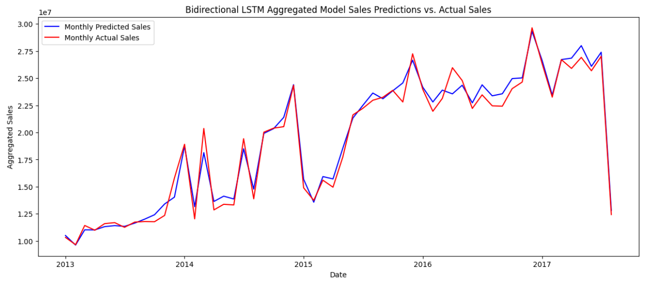

Figure 9: Bidirectional LSTM Model Sales Predictions vs. Actual Sales aggregated monthly

C GRU MODEL CHARTS

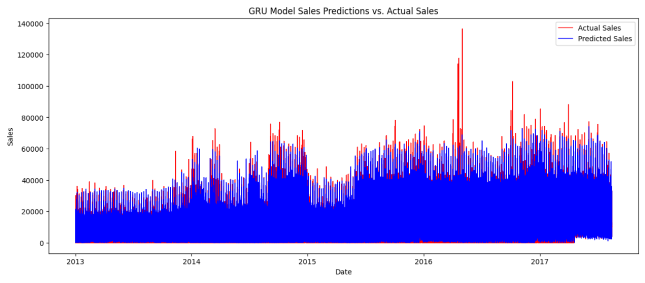

Figure 10: GRU Model Sales Predictions vs. Actual Sales

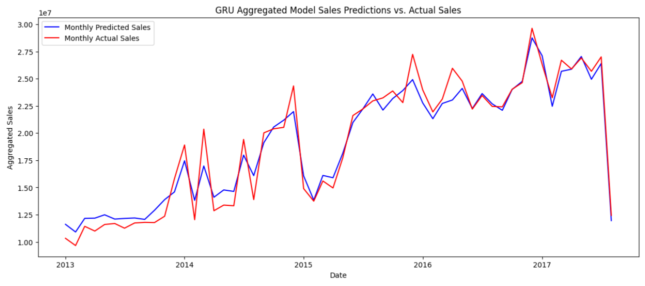

Figure 11: GRU Model Sales Predictions vs. Actual Sales aggregated monthly