Convolutional Approach to House Price Regression

1st Tim Heydrich
Department of Computer Science
Lakehead University
Tim Heydrich, 1121117
February 13, 2020

***Abstract*—Regression problems are very common and with classification make up the two main parts of machine learning. This makes it no surprise that Convolutional Neural Networks (CNNs) would also be applied to this area. In this paper several a general overview of CNNs is given. Furthermore, some CNN models are proposed to solve the issue of house price estimation.**

I. Introduction

Machine learning tasks can usually be categorized into two main groups, classification and regression. Classification describes the more straight forward approach of determining the class of a certain input. These classifications can be very simple binary classification of for example cats and dogs or more complex multi-class classification with multiple possible categories.

Regression differs from classification in the way that the end goal usually represents any real value and is not limited to a certain number of classes. Regression usually revolves around predictions, for example stock predictions or sales predictions.

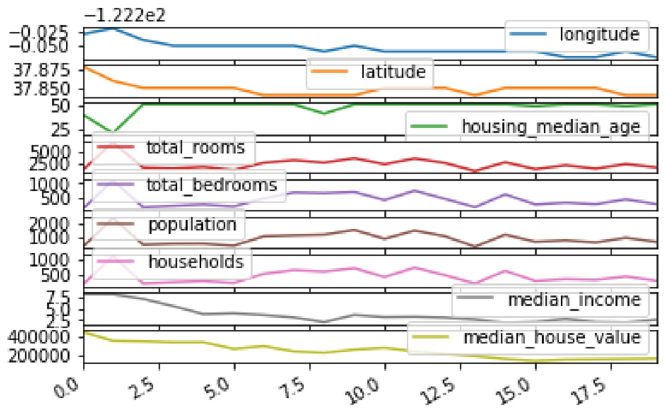

Fig. 1. Plot of the data entries in the housing.csv dataset

In this paper the focus problem is the house price prediction which is a regression problem. The dataset used in this paper is a modified version of the California Housing Prices dataset which is retrievable through the GitHub [1] of 'Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow'. The dataset consists of a table of different properties that can all influence the price of a house.

II. Background

A. Convolutional Neural Networks

Convolutional Neural Networks [2] have been around for several year, however, in recent years they gained a lot of popularity due to their effectiveness especially with computer vision tasks. Development of more powerful graphic processing units (GPUs) made this growth possible due to their inherent ability to work in parallel as well as their strength when it comes to computing floating point units. Stronger GPUs drastically reduced the training time needed to teach a neural network. This meant that larger and deeper network structures were possible without needing days to complete the training of the architecture.

The strength of CNNs for computer vision tasks comes from the way the so called convolutional layers work. Each convolutional layer consists of multiple kernels, or filters, of a given size, the most commonly used sizes are 3x3, 5x5, 6x6. These kernels work similar to other kernel applications in computer vision, for example the edge detectors or blurs. The general kernel computation [3] is the same as for other kernel and traversal of the image is performed with a previously determined step size. Unlike other kernel applications the CNN kernels do not produce another image of the same size, the kernels do not overlap the edges of the data. In simpler terms, the size is reduced to the number of possible locations a kernel can be placed on the data without overlapping. As an example suppose a give 5x5 kernel is applied to a 32x32 image then the result after the convolution is a 28x28 'image'. This kernel application clearly results in a reduction of dimensionality, feature extraction. Continuing the use of images as an example, images usually wither consist of a single channel, grey images, or three channels, colour images. Utilizing multiple kernels during each convolution layer not only reduces the dimensionality but also increases the channel count to the number of filters used in the layer. Each filter produces it's own convolved 'image' which are then further processed in the next layers. The use of multiple layers of convolution gives lots of feature maps with high and low level features extracted from the data.

Convolutional neural networks do however have a drawback when it comes to tabular data such as price predictions or EEG output [5]. There are several issues when using CNNs with tabular data. The first one being that most tabular data does not consist of as many columns as an image would,

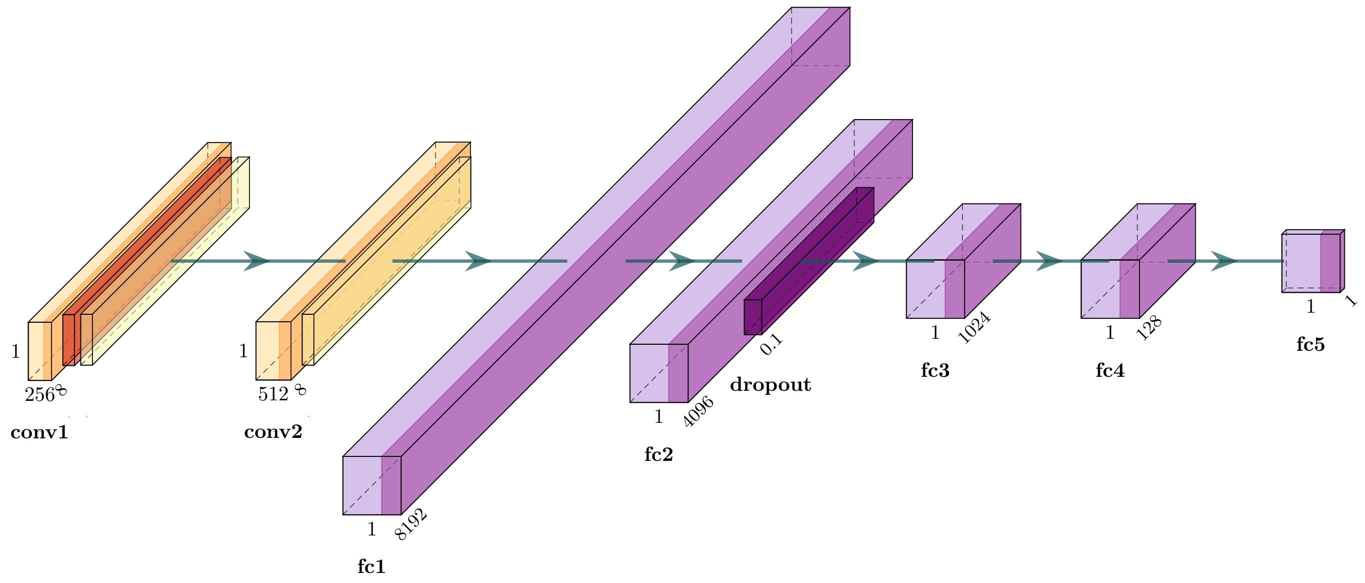

Fig. 2. Final Model Architecture Created with a modified version of [4]

this severely limits the ability to extract key features as the dimensionality is already very low. Another reason is the fact that for tabular data two (2) dimensional convolution, which was explained above, does not work as only a single row is relevant for the target value. This means that one (1) dimensional convolution has to used on tabular data. One dimensional filters, for example 3x1, are not nearly as effective as two or multi dimensional filters. The combination of the two issues mentioned above limits the effectiveness of the convolution to extract enough and important features from the tabular data.

B. *Common Approaches for Tabular Data*

Tabular data does not work well with CNNs, reasons given above II-A. There are other approaches that are more commonly used. The most straight forward being a simple multi layer Fully Connected Neural Networks (FCNNs). FCNNs are appropriate for quick and easy prototyping or data exploration they often do not yield high accuracy. Other possible approaches include the use of a Random Forest [6] or Support Vector Machine (SVM) [7]. The currently most successful approach used is the so call eXtreme Gradient Boost (XGBoost) [8] which is a fine tuned application of gradient boosting [9].

Gradient boosting is used to model complex nonlinear loss functions. The basic idea is that complex problems can be modeled through the use of many simple models. Complex nonlinear problems can usually not be modeled by simpler models such as for example a linear model, the error produced is too high. Continuing the use of a linear base model as an example, gradient boosting utilizes the difference between the prediction and the actual data. This difference, the error, is used then used to train another linear model in an attempt to model the error produced by the first. These two models are then added together, the resulting model performs better than the single linear model. This step is then repeated for several intervals. This way a complex problem is being modeled through the use of many simpler models.

III. Model Architecture

A. *Basic Architecture*

The design architecture of this model, with the challenges mentioned in II-A in mined, posed difficult. The basic model that was first during the lab session [10] consisted of a six (6) layer CNN. The first layer was a one dimensional convolution layer with 64 channels and a kernel size of one. This was followed by a Max-Pooling layer and another convolution layer with the same kernel size but 128 filters. These convolution layers were then followed by the flatten layer and two dense layers. The first dense layer consisted of 64 neurons and the second was the output layer, meaning it only had a single neuron. This model was able to learn from the given data, the R^2 Score proved not feasible as it was reaching ~0.25 on a good run.

B. *Intermediate Steps and the Challenges*

The initial 'improvements' to the basic model did not prove successful. Adding more convolution layers or other types of layers such as batch normalization resulted in no success. The model would not improve during the training process and would start and finish with an R^2 Score of ~(-3) which indicates that taking the mean of the data would have been about 300% more effective than the model. The issue of not learning could be related to the fact that convolution on tabular data does not work properly this needs to be explored in greater detail.

Improvement attempts staggered after the failure and the time investment into understanding the issue at hand. Due to the information that single dimension convolution is not very effective the adding of additional fully connected, dense, layers seemed logical. This did increase the accuracy of the basic model, however, it was still not at a satisfactory level.

C. Final Model

The final model (fig.2) only consists of two convolution layers as well as a single max-pooling layer. In addition, two batch normalization layers were added as well as four additional fully connected layers. This model still struggled with the same problems that the previous models III-B. The addition of the batch-normalization was not part of the initial mode, it was added later on as it increased the accuracy significantly. The main issue at hand was solved through the normalization of the training data. This allowed the more complex models to learn and improve. The initial normalization was in the range of 0-1 this was then later revised to 0-50 as it improved the accuracy by a few percent due to the fact that the values were not as close together as in the range of 0-1.

The final model architecture is a convolutional layer with 256 kernels of size one followed by a max-pooling with a pool size of 1 and a batch normalization. This is followed by the second convolution layer and the second batch normalization. The key component of the model are the fully connected layers. These layers decrease in size starting with 8192 nodes followed by 4096 and 1024. The layer before the output layer is of size 128 followed by the output layer which contains a single neuron. After the first two fully connected layers a dropout layer with a factor of 0.1 was added to due some mild overfitting issues in the model. The factor of 0.1 was sufficient to prevent overfitting as well as larger values decreased the accuracy of the model.

D. XGBoost with CNN

After performing initial testing with the final model III-C and observing how well XGBoost performs on regression problems a combination of the two came to mind to boost the performance of the CNN model. The researchers in [11] used a combination of CNN and XGBoost to enhance their model.

The basic concept of the combination model is that only the feature extraction of the CNN is used and that the actual prediction is performed by the XGBoost. This means that the CNN is trained and the output of pretrained CNNs feature layers are used as input for the XGBoost to train it on the data. The final model would be a combination where the data is given to the CNN and the feature output is fed to the XGBoost which then makes the prediction.

Combining the two models proved challenging due to the lack of proper documentation as well as unfamiliar PyTorch environment. A combination of the initial basic model III-A as well as the final model III-C were performed and trained.

	MSE	L1Loss	R2Score
Linear Regression	54.26	N/A	0.60
Basic Model (BM)	102.22	7.59	0.24
(BM) + 1 Convolution & Max-Pooling	119.83	8.31	0.13
(BM) + Batch Normalization	516.31	19.43	-2.80
Final Model w/o Batch Normalization	56.26	5.19	0.60
Final Model w/ Batch Normalization	**41.24**	**4.35**	**0.70**
XGBoost [11]	N/A	N/A	0.82
BM XGBoost	N/A	N/A	-1.45
Final Model XGBoost	N/A	N/A	-1.35

TABLE I
COMPARISON OF VARIOUS MODELS TESTED ON THE DATASET

IV. Comparison and Evaluation

A. Comparison

The comparisons performed in this section are performed on several different in between steps of the model development as well as other non CNN models used for regression. The dataset used for the training and evaluation has been preprocessed with normalization in the range of 0-50. Each model was trained and tested three (3) times and the average of the three tests were recorded. The results can be seen in table I. It needs to be noted that an increased number of epochs for the '(BM) + 1 Convolution Max-Pooling' resulted in a maximum R^2 Score of approximately 0.60

B. Evaluation

The comparison reveals that the final model III-C had the best results out of the convolution models with a R^2 Score of 0.70. The XGBoost results on the same dataset which were retrieved from [12] did however perform better with a R^2 Score of 0.82. Due to that success the combination models were thought of. The combination models did not perform very well with a R^2 Score of -1.45 and -1.35. Despite the successes of combination models such as [10] that approach did not perform well on this dataset. There are a couple reasons that could have caused this, the first being that the implementation was not performed properly. The second is that the convolution layers were not able to extract significant enough features for the XGBoost and that the extracted features simply 'confused' the XGBoost.

V. Conclusion and Future Work

This paper clearly supports the existing knowledge of issues with using convolution on tabular data. The final model proposed here was able to perform better than linear Regression models, however, it was unable to beat the XGBoost approach which is currently still the best option for this dataset. When using tabular data it is recommended that XGBoost is used instead of CNN approaches until a suitable way to apply CNN models to tabular data is found.

Future work in this area would revolve around this topic. There are some datasets where a reshaping of the data would make it possible to better apply convolutional approaches such as the EEG dataset where the data can be reshaped to represent a two dimensional matrix. Furthermore, approaches could include an adaption of the gradient boosting approach but for CNN models where multiple less complex models are added together. This would obviously almost form a deep neural network but it is a possible path to be explored.

References

[1] ageron, "handson-ml," https://github.com/ageron/handson-ml.

[2] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[3] V. Powell. Image kernels. [Online]. Available: http://setosa.io/ev/image-kernels/

[4] H. Iqbal, "Harisiqbal88/plotneuralnet v1.0.0," Dec. 2018. [Online]. Available: https://doi.org/10.5281/zenodo.2526396

[5] J. Bird. Eeg brainwave dataset: Feeling emotions. [Online]. Available: https://www.kaggle.com/birdy654/eeg-brainwave-dataset-feeling-emotions

[6] A. Liaw, M. Wiener *et al.*, "Classification and regression by randomforest," *R news*, vol. 2, no. 3, pp. 18–22, 2002.

[7] T. Evgeniou and M. Pontil, "Support vector machines: Theory and applications," vol. 2049, 01 2001, pp. 249–257.

[8] T. Chen and C. Guestrin, "Xgboost," *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*, 2016. [Online]. Available: http://dx.doi.org/10.1145/2939672.2939785

[9] J. H. Friedman, "Stochastic gradient boosting," *Computational statistics & data analysis*, vol. 38, no. 4, pp. 367–378, 2002.

[10] A. Fisher, "2 multilayer 1d conv network in pytorch."

[11] L. Li, R. Situ, J. Gao, Z. Yang, and W. Liu, "A hybrid model combining convolutional neural network with xgboost for predicting social media popularity," in *Proceedings of the 25th ACM international conference on Multimedia*, 2017, pp. 1912–1917.

[12] J. O. Ferris. California housing prices (a linear regression and a xgboost). [Online]. Available: https://www.kaggle.com/joseconomy/california-housing-prices-regression-with-xgboost