Implementing a Convolution Neural Network using linear regression model

Prerak Jayeshkumar Patel
Department of Computer Science
Lakehead University
Thunder Bay, Canada
Student Id: 1117675

***Abstract*—This assignment contains the implementation of one dimensional convolution neural network. The final score for performance analysis are 0.6611 for R2 score and 47587.92 for Loss score.**

I. Introduction

For the prediction of any value from the analysis of the previously collected dataset can be easily implemented using a convolutional neural network. Here, the dataset which is taken into consideration is the California Housing dataset. This dataset is collective information about all the block groups in California in 1990 collected using that year's census. The source of this dataset is StatLib, a dataset archive. There is a total of 20640 samples with 10 features. There is an addition of the last feature i.e. ocean proximity to experiment with categorical data. With the motive of predicting the median house value from the remaining nine features namely, longitude, latitude, housing median age, the total number of rooms, the total number of bedrooms, population, number of households, and median income, the solution was found implementing a one-dimensional convolution (Conv1D), neural network model. Many parameters like type of optimizer, number of kernels, number of epochs, size of the filter, number of hidden neurons, number of trainable parameters and more are responsible for the final performance of the model. Selecting the perfect parameters that in a way helps in assigning the perfect weights to the network is a task involving vigorous trial and error.

II. Literature Review

For the exploration of the possible previous work done on this dataset a paper by Xueheng had a good outcome [1]. The paper discussed all the possible models possible on the dataset and which included CNN, LSTM, SVM models for regression model. They observed that the parameter selection and network modeling play a good role in performance of the model.

III. Proposed Model

The proposed model has a single-layered one-dimensional convolutional neural network. The input when fad to the input layer, an 8x15 output is generated, as the batch size is set to 15. The output is then given to a max pool layer. The output of the max pool is the same as the input. This is used to overcome the problem of overfitting. The Output of the max pool layer is then fad to a 1D Convolution layer with 17 number of neurons and kernel size equals 1. This generates a 15x17 output. After that, the output is flattened and converted into the form which can be given to the fully connected linear layer and a single number as output is generated.Special note must be taken of the use of Relu function before every pooling layer. Now, for the training process, there is one use of optimizer. The optimizer implements an RMSprop Algorithm proposed by G. Hinton. The learning rate is 0.001 with a momentum of 0.9. The number of epochs used for testing and training is 100. The final output from this model is showing some promising results which we will discuss in the future part of the report.

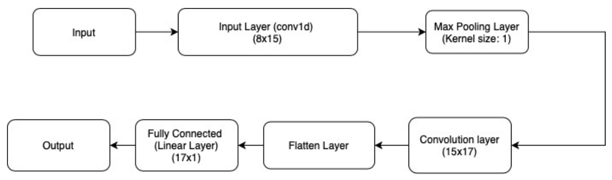
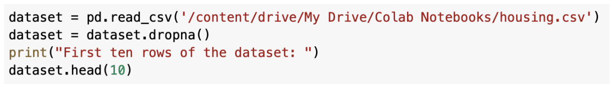

Input
Input Layer (conv1d) (8x15)
Max Pooling Layer (Kernel size: 1)
Convolution layer (15x17)
Flatten Layer
Fully Connected (Linear Layer) (17x1)
Output

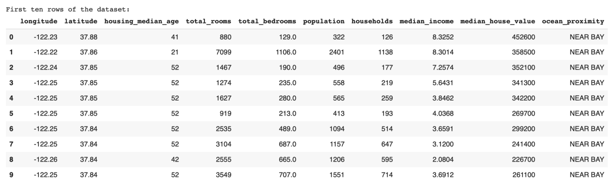
Fig. 1. Block diagram of proposed model

The basic flow of the implantation of this can be seen in the block diagram of the model. The input data is loaded into a panda data frame from the csv file. The dronna() function of panda library is used to drop the rows with incomplete fields. The first ten record are showed by using the head() attribute of panda library.



```
dataset = pd.read_csv('/content/drive/My Drive/Colab Notebooks/housing.csv')
dataset = dataset.dropna()
print("First ten rows of the dataset: ")
dataset.head(10)
```

Fig. 2. Code snippet 1



First ten rows of the dataset:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41	880	129.0	322	126	8.3252	452600	NEAR BAY
1	-122.22	37.86	21	7099	1106.0	2401	1138	8.3014	358500	NEAR BAY
2	-122.24	37.85	52	1467	190.0	496	177	7.2574	352100	NEAR BAY
3	-122.25	37.85	52	1274	235.0	558	219	5.6431	341300	NEAR BAY
4	-122.25	37.85	52	1627	280.0	565	259	3.8462	342200	NEAR BAY
5	-122.25	37.85	52	919	213.0	413	193	4.0368	269700	NEAR BAY
6	-122.25	37.84	52	2535	489.0	1094	514	3.6591	299200	NEAR BAY
7	-122.25	37.84	52	3104	687.0	1157	647	3.1200	241400	NEAR BAY
8	-122.26	37.84	42	2555	665.0	1206	595	2.0804	226700	NEAR BAY
9	-122.25	37.84	52	3549	707.0	1551	714	3.6912	261100	NEAR BAY

Fig. 3. First 10 samples from dataset

By using the Matplot library's function of subplot we plotted the graph of each feature for data visualization.

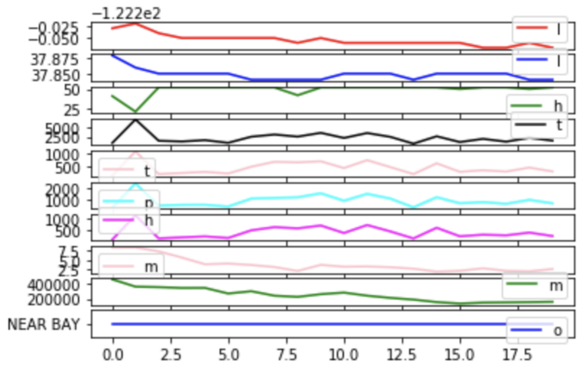
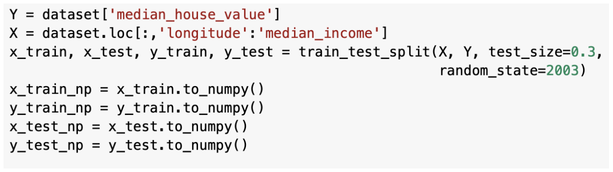


Fig. 4. Data visualization

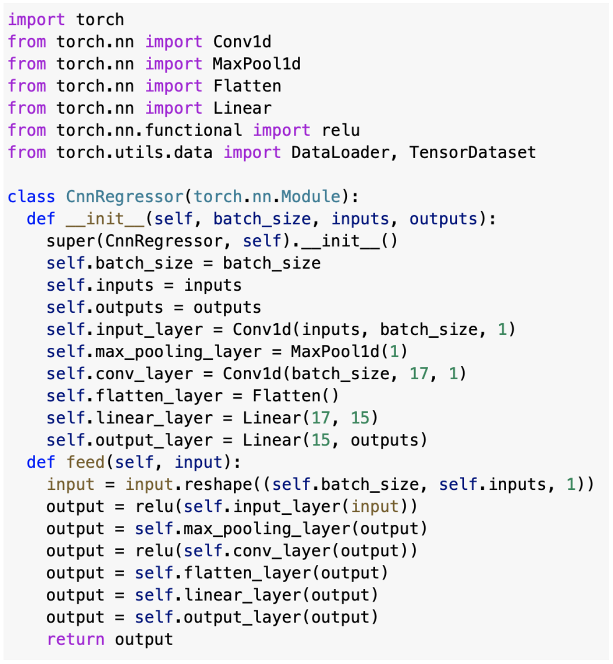
To separate out the dataset for predicting median house value and to test them on the model we made use of train test split of sklearn.model selection library.



```
Y = dataset['median_house_value']
X = dataset.loc[:,'longitude':'median_income']
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.3,
                                                    random_state=2003)
x_train_np = x_train.to_numpy()
y_train_np = y_train.to_numpy()
x_test_np = x_test.to_numpy()
y_test_np = y_test.to_numpy()
```

Fig. 5. Code snippet 2

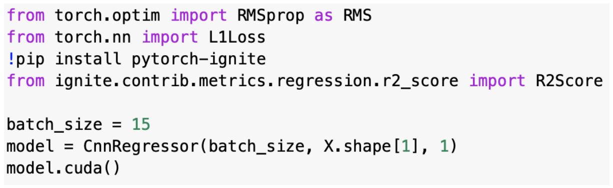
Imported class of CNN regressor must be subclass of torch module. So we initialized the CnnRegressor class as given below.



```
import torch
from torch.nn import Conv1d
from torch.nn import MaxPool1d
from torch.nn import Flatten
from torch.nn import Linear
from torch.nn.functional import relu
from torch.utils.data import DataLoader, TensorDataset

class CnnRegressor(torch.nn.Module):
  def __init__(self, batch_size, inputs, outputs):
    super(CnnRegressor, self).__init__()
    self.batch_size = batch_size
    self.inputs = inputs
    self.outputs = outputs
    self.input_layer = Conv1d(inputs, batch_size, 1)
    self.max_pooling_layer = MaxPool1d(1)
    self.conv_layer = Conv1d(batch_size, 17, 1)
    self.flatten_layer = Flatten()
    self.linear_layer = Linear(17, 15)
    self.output_layer = Linear(15, outputs)
  def feed(self, input):
    input = input.reshape((self.batch_size, self.inputs, 1))
    output = relu(self.input_layer(input))
    output = self.max_pooling_layer(output)
    output = relu(self.conv_layer(output))
    output = self.flatten_layer(output)
    output = self.linear_layer(output)
    output = self.output_layer(output)
    return output
```

Fig. 6. Code snippet 3

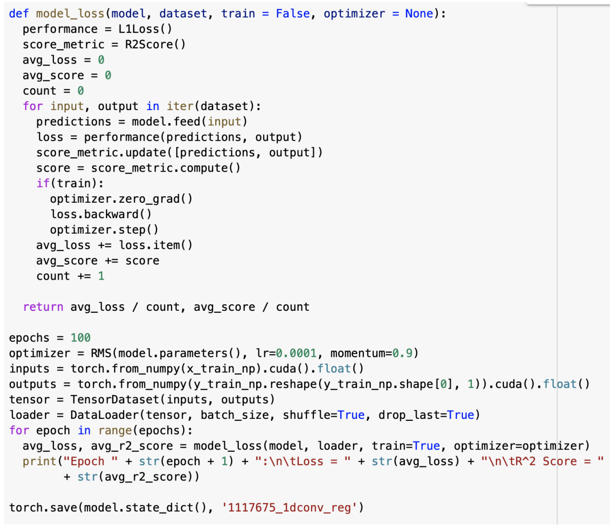
The library of optiu present in torch to use the RMSprop algorithm is imported. Also, for L1 Loss and R2 score's libraries are imported. The model is initialized by the batch size. And the cuda function is called to run this model on GPU.



```
from torch.optim import RMSprop as RMS
from torch.nn import L1Loss
!pip install pytorch-ignite
from ignite.contrib.metrics.regression.r2_score import R2Score

batch_size = 15
model = CnnRegressor(batch_size, X.shape[1], 1)
model.cuda()
```

Fig. 7. Code snippet 4

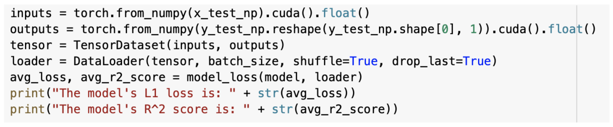
The training phase is explained in implemented and 100 epochs are set. The learning rate is 0.001 with momentum of 0.9 in the optimizer RMS. The trained model is stored in a file at the end.



```
def model_loss(model, dataset, train = False, optimizer = None):
  performance = L1Loss()
  score_metric = R2Score()
  avg_loss = 0
  avg_score = 0
  count = 0
  for input, output in iter(dataset):
    predictions = model.feed(input)
    loss = performance(predictions, output)
    score_metric.update([predictions, output])
    score = score_metric.compute()
    if(train):
      optimizer.zero_grad()
      loss.backward()
      optimizer.step()
    avg_loss += loss.item()
    avg_score += score
    count += 1

  return avg_loss / count, avg_score / count

epochs = 100
optimizer = RMS(model.parameters(), lr=0.0001, momentum=0.9)
inputs = torch.from_numpy(x_train_np).cuda().float()
outputs = torch.from_numpy(y_train_np.reshape(y_train_np.shape[0], 1)).cuda().float()
tensor = TensorDataset(inputs, outputs)
loader = DataLoader(tensor, batch_size, shuffle=True, drop_last=True)
for epoch in range(epochs):
  avg_loss, avg_r2_score = model_loss(model, loader, train=True, optimizer=optimizer)
  print("Epoch " + str(epoch + 1) + ":\n\tLoss = " + str(avg_loss) + "\n\tR^2 Score = "
        + str(avg_r2_score))

torch.save(model.state_dict(), '1117675_1dconv_reg')
```

Fig. 8. Code snippet 5

The testing is done on the testing data and final L1 loss and R2 score are calculated.



```
inputs = torch.from_numpy(x_test_np).cuda().float()
outputs = torch.from_numpy(y_test_np.reshape(y_test_np.shape[0], 1)).cuda().float()
tensor = TensorDataset(inputs, outputs)
loader = DataLoader(tensor, batch_size, shuffle=True, drop_last=True)
avg_loss, avg_r2_score = model_loss(model, loader)
print("The model's L1 loss is: " + str(avg_loss))
print("The model's R^2 score is: " + str(avg_r2_score))
```

Fig. 9. Code snippet 6

IV. Conclusion

The proposed CNN model has shown a prominent performance of 0.6611 R2 Score and 47587.92 of Loss score. The time for training is also acceptable. The prediction of house value can be done using this model.

REFERENCES

[1] Qiu, Xueheng, Le Zhang, Ye Ren, Ponnuthurai N. Suganthan, and Gehan Amaratunga. ”Ensemble deep learning for regression and time series forecasting.” In 2014 IEEE symposium on computational intelligence in ensemble learning (CIEL), pp. 1-6. IEEE, 2014.