Driven by Data

Predicting Car Prices with a Hybrid Dense and CNN Model

Ian Charamuga
iancharamuga@csus.edu
California State University, Sacramento
Sacramento, California, USA

Phuc Dinh
phucvandinh@csus.edu
California State University, Sacramento
Sacramento, California, USA

Jacob Sullivan
jacobdsullivan@csus.edu
California State University, Sacramento
Sacramento, California, USA

Amir Talakoob
amirtalakoob@csus.edu
California State University, Sacramento
Sacramento, California, USA

ABSTRACT

In today's world, despite the development of public transport systems, having a personal vehicle is hugely beneficial in most cities. Like many others, university students can significantly benefit from owning a car; however, such a purchase is an important financial decision. Considering used vehicles, without having basic knowledge regarding the price estimation of what the individual is interested in buying, the risk of not finding an optimal option or even paying extra increases. At this point, the satisfactory knowledge will be the estimated price of the car. In different instances, the individual might have various or even limited information about the car they want to evaluate the price for. For example, they might only have a picture of the front of the car, or they have no photo and only some basic specs such as mileage, color, fuel type, etc. Hence, our team developed different models and approaches to predict the price using different inputs. Given our motivation and availability of the data, we mainly focused on used cars and developed multiple models with various input types. Some of the models are as follows: a model that only uses four images, from the front, right side, left side, and rear of the car; a model that uses only text-based data; a model that uses a combination of the previously mentioned models, and a model that uses transfer learning. For the training and development of these models, we used the DVM-CAR data set that consisted of multiple data tables.

ACM Reference Format:
Ian Charamuga, Phuc Dinh, Jacob Sullivan, and Amir Talakoob. 2023. Driven by Data: Predicting Car Prices with a Hybrid Dense and CNN Model. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (CSC 180 - CSUS).* ACM, New York, NY, USA, 8 pages. https://doi.org/XXXXXXX.XXXXXXX

CSC 180 - CSUS, November 27, 2023, Sacramento, CA
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-CSUS-180/18/06
https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

1.1 Problem

What we are focused on for this project is trying to predict the price of a used car. Owning a car is more vital than ever, especially ever since COVID-19 hit, so it would be nice to have an idea of what an individual would need to pay to get a used car, given either the specs or photos.

1.2 Approach

To achieve this, we start by observing our data tables and joining them to find the best input variations for our to-be-developed models. After an extensive and time-consuming data preprocessing (the image data set was over 13 GB), we came up with the following input combinations: a) images from the front of the car, b) images from the front, left, right, and back sides of the car, c) only text-based data using both categorical and numerical variables d) a combination of the b and c, e) transfer learning. For most of these models, we used the Functional API, and our output was the price. [2]

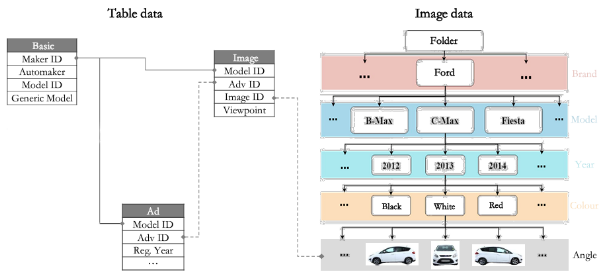


Figure 1: Table Connections

1.3 Contributions

1.3.1 Ian Charamuga:

- Design and implementation of the regression models.
- Creating the skeleton for some of the models.
- Using transfer learning for some parts of the project.
- Parameter tuning for models.

1.3.2 *Phuc Dinh:*

- Data preprocessing and setting up the skeleton of the notebooks.
- Using transfer learning for some parts of the project.
- Creating the skeleton for some of the models.
- Creating the base for the CNN model.

1.3.3 *Jacob Sullivan:*

- Data preprocessing and setting up the skeleton of the notebooks.
- Creating the skeleton for some of the models.
- Parameter tuning for models.
- Creating the base for the Dense model.

1.3.4 *Amir Talakoob:*

- Design and implementation of the four image paths in functional API
- Data preprocessing (images table) and creating the skeleton for some of the models.
- Parameter tuning for models.
- Experimenting with RNN (LSTM) models (unsuccessful due to the nature of the data set).

2 PROBLEM FORMULATION

In order to train a model to predict used car prices, we are using the DVM-CAR dataset. This dataset contains prices of 10 years of car sales data in the UK. In addition, there are multiple accompanying images for each car from different angles. We construct multiple neural network models in Python using Tensorflow and associated data science libraries (pandas, numpy). These models were run in the Google Colab cloud environment. We created multiple models using different combinations of this data (sales data, images, and both) and compared the results of each model type. The following are all the input variables that we used: 'Adv_year', 'Adv_month', 'Color', 'Reg_year', 'Bodytype', 'Runned_Miles', 'Engin_size', 'Gearbox', 'Fuel_type', 'Price', 'Seat_num', 'Door_num'. We also tried different ways to treat the inputs. Our only output was the estimated price of the car.

3 SYSTEM/ALGORITHM DESIGN

3.1 System Architecture

We created multiple Dense, CNN, and Hybrid (Dense + CNN) models with different hyper-parameters, neuron counts, number of layers, and activation functions. We also developed models using transfer learning methods, specifically VGG16. All models output a dollar amount prediction for a used car, given its input data. In short, for training all of our models, we connected the tables, removed the outliers, handled the image data (since each car (make, model, year, color) must have exactly 4 images), removed the unwanted columns, one-hot encoded the categorical input features, normalized the numeric features, separated input and output, split the train and test, and then began the training the model that was developed in functional API.

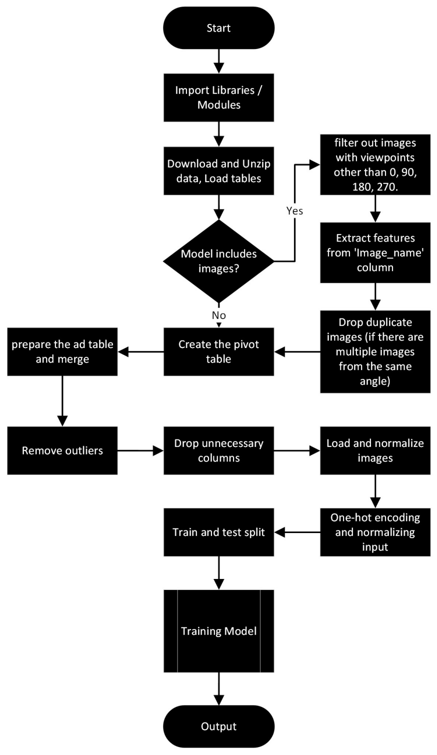

Start
Import Libraries / Modules
Download and Unzip data, Load tables
Model includes images?
Yes
No
filter out images with viewpoints other than 0, 90, 180, 270.
Extract features from 'Image_name' column
Drop duplicate images (if there are multiple images from the same angle)
Create the pivot table
prepare the ad table and merge
Remove outliers
Drop unnecessary columns
Load and normalize images
One-hot encoding and normalizing input
Train and test split
Training Model
Output

Figure 2: Flowchart of our architecture

3.2 Dense Model

3.2.1 *Algorithm Description.* Algorithm A

```
  1. Input ad data into the input layer.
2. Process data through hidden layers with
   100, 50, 25, and 10 neurons, respectively.
3. End model with single output neuron.
```

3.2.2 *Algorithm Description.* Algorithm B

```
  1. Input ad data into the input layer.
2. Process data through hidden layers with
    200, 150, 60, 25, and 10 neurons respectively.
3. End model with single output neuron.
```

3.2.3 *Algorithm Description.* Algorithm C

```
  1. Input ad data into the input layer.
2. Process data through hidden layers with
 200, 150, 60, 25, and 10 neurons, respectively with
    the first layer using tanh.
3. End model with single output neuron.
```

3.3 CNN Model

3.3.1 *Algorithm Description.* Algorithm A

```
1. Input image data into the input layer
   (shape: (64, 64, 3)).
2. Process each of the 4 images through one 32-filter
   VGG block
      a. 2 convolution layers (32 filters,
          kernel size (3, 3))
      b. 1 max pooling layer (pool size (2, 2))
      c. 1 dropout layer (20% of neurons hidden)
   3. Flatten data after each of the 4 sets of VGG blocks
      with a flatten layer.
   4. Combine all layers into one.
   5. Process data through 1 hidden layer with 128 neurons.
   6. End model with single output neuron.
```

3.3.2 *Algorithm Description.* Algorithm B

```
   1. Input image data into the input layer
     (shape: (64, 64, 3)).
   2. Process data through one 32-filter VGG block and
      one 64-filter VGG block.
      a. 2 convolution layers (32 and 64 filters,
         kernel size (3, 3)).
      b. 1 maxpooling layer (pool size (2, 2)).
      c. 1 dropout layer (20% of neurons hidden)
   3. Flatten data after each of the 4 sets of VGG blocks
      with a flatten layer.
   4. Combine all layers into one.
   5. Process data through 1 hidden layer with 128 neurons.
   6. End model with single output neuron.
```

3.4 Hybrid Model

3.4.1 *Algorithm Description.* Algorithm A

```
  1. Process ad data.
      a. Input ad data into the input layer.
    b. Process data through hidden layers with
       100, 50, 25, and 10 neurons, respectively.
  2. Process image data.
   a. Process each of the 4 images through one 32-filter
      VGG block.
      i. 2 convolution layers (32 filters,
          kernel size (3, 3)).
      ii. 1 max pooling layer (pool size (2, 2)).
      iii. 1 dropout layer (20% of neurons hidden).
   b. Flatten data after each of the 4 sets of VGG blocks
      with a flatten layer.
  3. Merge all layers into one.
  4. Process data through 1 hidden layer with 128 neurons.
  5. End model with single output neuron.
```

3.4.2 *Algorithm Description.* Algorithm B

```
    1. Process ad data
        a. Input as data into the input layer.
      b. Process data through hidden layers with
         200, 150, 60, 25, and 10 neurons respectively.
    2. Process image data
     a. Process each of the 4 images through one 32-filter
        and one 64-filter VGG block.
        i. 2 convolution layers (32 and 64 filters,
          kernel size (3, 3)).
        ii. 1 max pooling layer (pool size (2, 2)).
        iii. 1 dropout layer (20% of neurons hidden).
     b. Flatten data after each of the 4 sets of VGG blocks
        with a flatten layer.
    3. Merge all layers into one.
    4. Process data through 1 hidden layer with 128 neurons.
    5. End model with single output neuron.
```

3.4.3 *Algorithm Description.* Algorithm C

```
    1. Process ad data
        a. Input as data into the input layer.
      b. Process data through hidden layers with
         100, 50, 25, and 10 neurons, respectively.
    2. Process image data
     a. Process only front images through 32, 64, and 128 kernels
        Conv2D layers with input (64, 64, 3)
         i. 2 convolution layers with 32 kernels,
         kernel size (3, 3)).
         ii. 1 max pooling layer (pool size (2, 2)).
         iii. 2 convolution layers with 64 kernels,
         kernel size (3, 3)).
         iiii. 1 max pooling layer (pool size (2, 2)).
         iiiii. 2 convolution layers with 64 kernels,
         kernel size (3, 3)).
        iiiiii. 1 max pooling layer (pool size (2, 2)).
       b. Flattern layer.
    3. Merge all layers into one.
    4. Process data through 1 hidden layer with 128 neurons.
    5. End model with single output neuron.
```

3.4.4 *Algorithm Description.* Algorithm D

```
    1. Process ad data
        a. Input as data into the input layer.
      b. Process data through hidden layers with
         100, 50, 25, and 10 neurons, respectively.
    2. Process image data
       a. Process 4 images (merged) through
       32, 64, 128 kernels Conv2D layers
       with input (128, 128, 3)
         i. 2 convolution layers with 32 kernels,
         kernel size (3, 3)).
         ii. 1 max pooling layer (pool size (2, 2)).
         iii. 2 convolution layers with 64 kernels,
         kernel size (3, 3)).
         iiii. 1 max pooling layer (pool size (2, 2)).
         iiiii. 2 convolution layers with 64 kernels,
         kernel size (3, 3)).
        iiiiii. 1 max pooling layer (pool size (2, 2)).
       b. Flattern layer.
    3. Merge all layers into one.
    4. Process data through 1 hidden layer with 128 neurons.
    5. End model with single output neuron.
```

3.5 Transfer Learning - VGG-16

3.5.1 *Algorithm Description.* Algorithm A

```
  1. Process ad data.
      a. Input ad data into the input layer.
    b. Process data through hidden layers with
       100, 50, 25, and 10 neurons, respectively.
  2. Process image data.
   a. Process only the front image through the VGG-16 model.
```

```
      i. 1 Flatten layer.
      ii. 2 Dense layer with 1024 and 256 neurons.
      iii. 1 dropout layer (20% of neurons hidden).
3. Merge all layers into one.
4. Process data through 1 hidden layer with 128 neurons.
5. End model with single output neuron.
```

3.5.2 Algorithm Description. Algorithm B

```
1. Process ad data.
      a. Input ad data into the input layer.
   b. Process data through hidden layers with
      100, 50, 25, and 10 neurons, respectively.
2. Process image data.
   a. Process 4 images (merged) through VGG-16 model.
      i. 1 Flatten layer.
      ii. 2 Dense layer with 1024 and 256 neurons.
      iii. 1 dropout layer (20% of neurons hidden).
3. Merge all layers into one.
4. Process data through 1 hidden layer with 128 neurons.
5. End model with single output neuron.
```

4 EXPERIMENTAL EVALUATION

4.1 Methodology

We used data on used car sales over a period of 10 years, including data such as miles driven, engine type, body type, and images of each car from multiple angles. We chose a random 25 percent of samples to be used as test data. The remaining 75 percent was used to train the model. This ensures the model is trained on sufficient data and evaluated on unseen data to assess its generalization performance. Besides train_test_split, EarlyStopping and Dropout layers are also used to prevent overfitting. We define the attributes that describe the used car as input and the price of the car as output. We then remove missing values and outliers, encode categorical variables, and normalize numeric attributes. Here, we also tried different ways to handle data to see its effects. We came up with different models along different hyperparameters to see how they performed on this particular dataset. We used the RMSE score and lift charts to compare the performance of each model.

4.2 Results

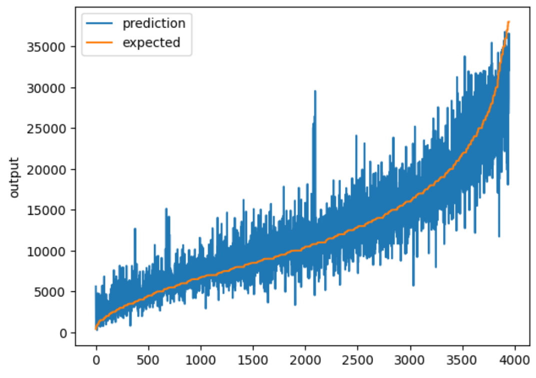


Figure 3: 100-50-25-10 Adam Dense Model

4.2.1 Dense Model - Algorithm A (Figure 3). Dense model A had an RMSE score of 2829.184

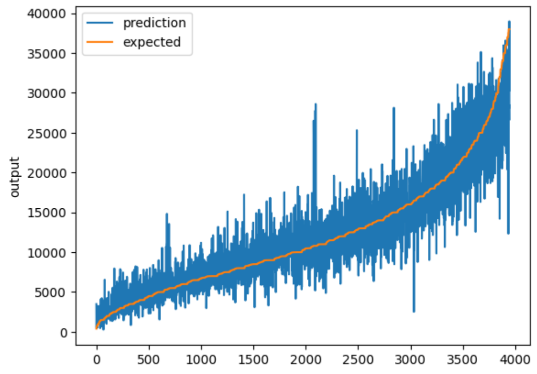


Figure 4: 200-150-60-25-10 Adam Dense Model

4.2.2 Dense Model - Algorithm B (Figure 4). Dense model B had an RMSE score of 2832.404

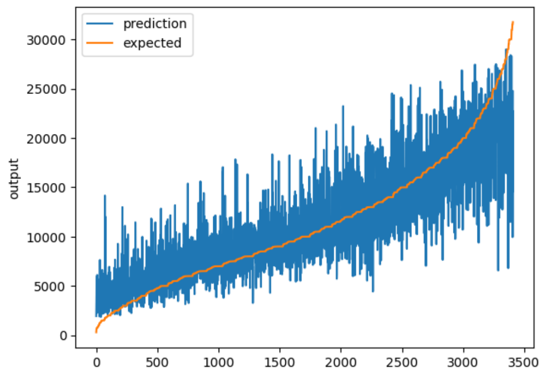


Figure 5: 200-150-60-25-10 Adam Dense Model

4.2.3 Dense Model - Algorithm B (Figure 5). Dense model B had an RMSE score of 3227.18 when normalizing Adv_year_month

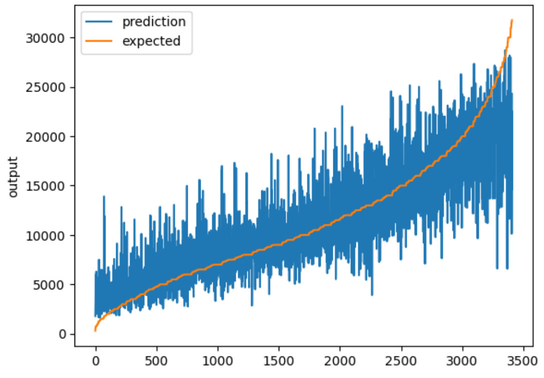


Figure 6: 200-150-60-25-10 Adam Dense Model

4.2.4 *Dense Model - Algorithm B (Figure 6).*
Dense model B had an RMSE score of 3228.335 when one-hot Adv_month and year

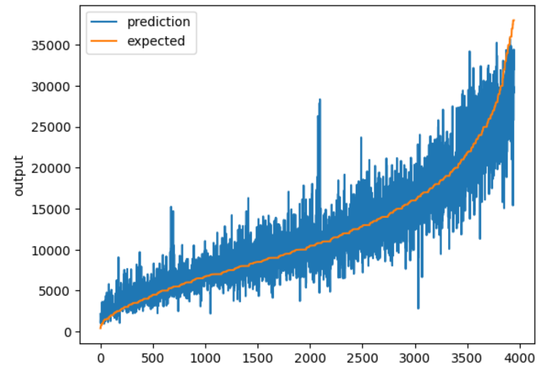


Figure 7: 200-150-60-25-10 Adam Tanh 1st Layer Dense Model

4.2.5 *Dense Model - Algorithm C (Figure 7).*
Dense model C had an RMSE score of 2850.730

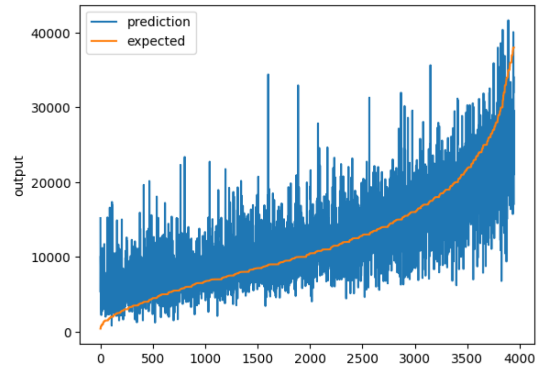


Figure 8: 1 VGG Block CNN Model

4.2.6 *CNN Model - Algorithm A (Figure 8).*
CNN model A had an RMSE score of 4565.287

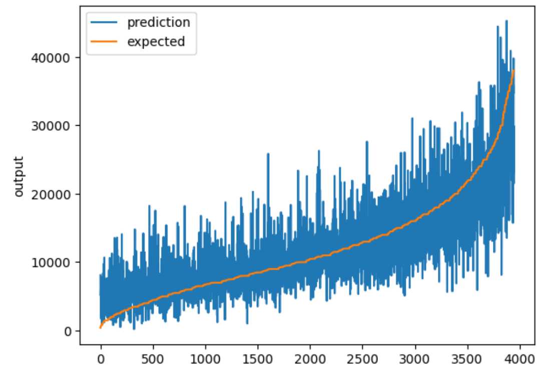


Figure 9: 2 VGG Blocks CNN Model

4.2.7 *CNN Model - Algorithm B(Figure 9).*
CNN model B had an RMSE score of 4218.435

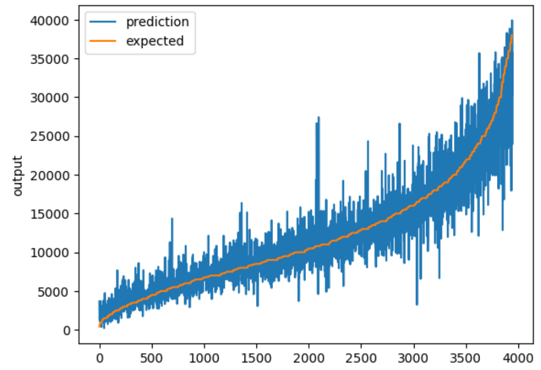


Figure 10: 100-50-25-10 Adam | 2 VGG Blocks Hybrid Model

4.2.8 *Hybrid Model - Algorithm A (Figure 10).*
Hybrid model A had an RMSE score of 2444.381

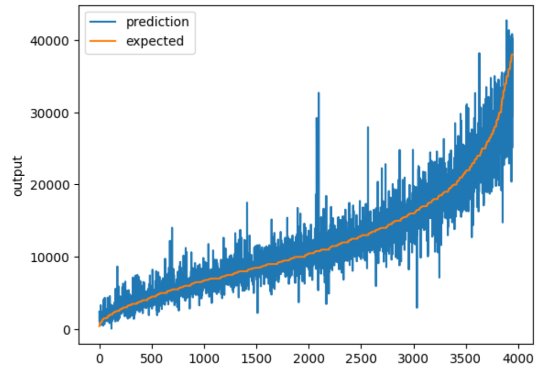


Figure 11: 200-150-60-25-10 Adam | 2 VGG Blocks Hybrid Model

4.2.9 *Hybrid Model - Algorithm B (Figure 11).*
Hybrid model B had an RMSE score of 2390.246

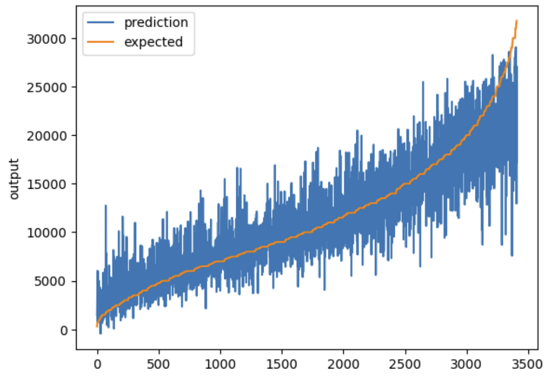


Figure 12: 100-50-25-10 Adam | front image Hybrid Model

4.2.10 Hybrid Model - Algorithm C (Figure 12). Hybrid model C had an RMSE score of 2878.447 when normalizing Adv_year_month

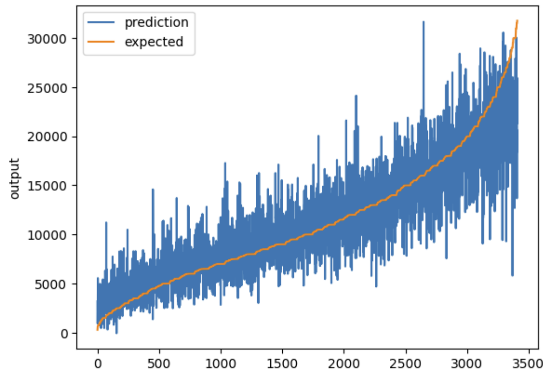


Figure 13: 100-50-25-10 Adam | front image Hybrid Model

4.2.11 Hybrid Model - Algorithm C (Figure 13). Hybrid model C had an RMSE score of 3032.744 when one-hot Adv_month and year

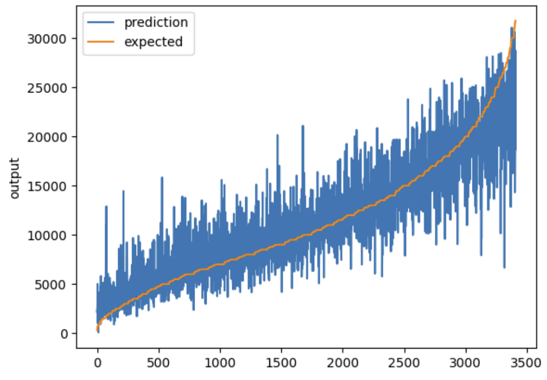


Figure 14: 100-50-25-10 Adam | 4 images Hybrid Model

4.2.12 Hybrid Model - Algorithm D (Figure 14). Hybrid model D had an RMSE score of 2864.516 when normalizing Adv_year_month

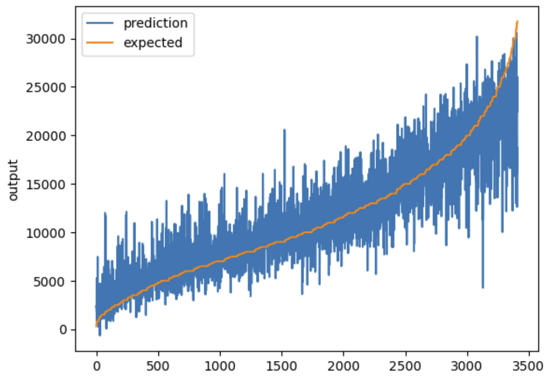


Figure 15: 100-50-25-10 Adam | 4 images Hybrid Model

4.2.13 Hybrid Model - Algorithm D (Figure 15). Hybrid model D had an RMSE score of 2880.030 when one-hot Adv_month and year

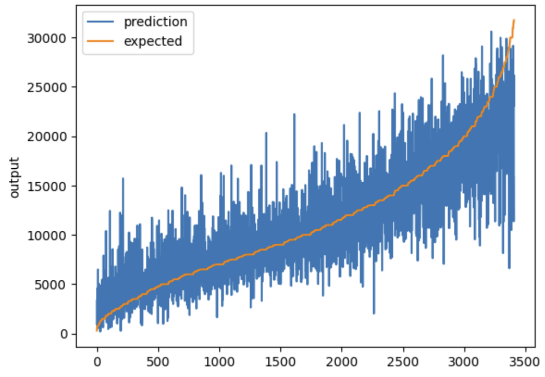


Figure 16: 100-50-25-10 Adam | front image VGG Model

4.2.14 Transfer Learning Model - Algorithm A (Figure 16). Transfer Learning Model A had an RMSE score of 3393.739 when normalizing Adv_year_month

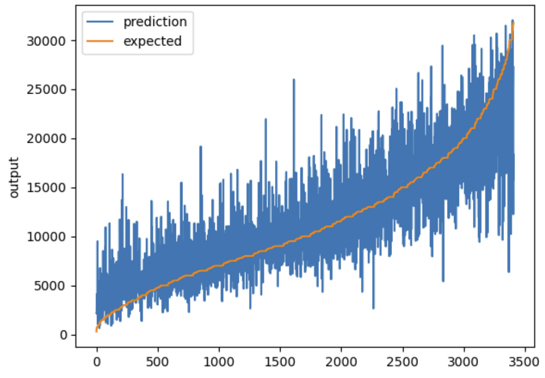


Figure 17: 100-50-25-10 Adam | front image VGG Model

4.2.15 Transfer Learning Model - Algorithm A (Figure 17). Transfer Learning Model A had an RMSE score of 3398.765 when one-hot Adv_month and year

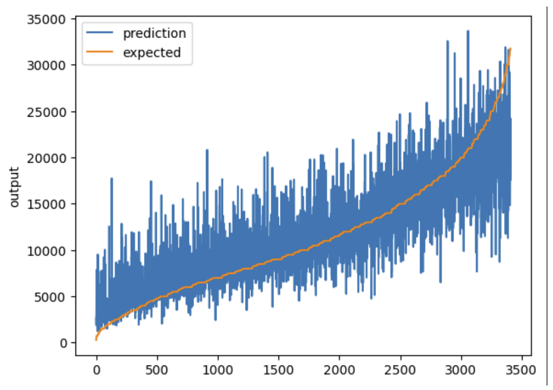


Figure 18: 100-50-25-10 Adam | 4 images VGG Model

4.2.16 Transfer Learning Model - Algorithm B (Figure 18). Transfer Learning Model B had an RMSE score of 3536.498 when normalizing Adv_year_month

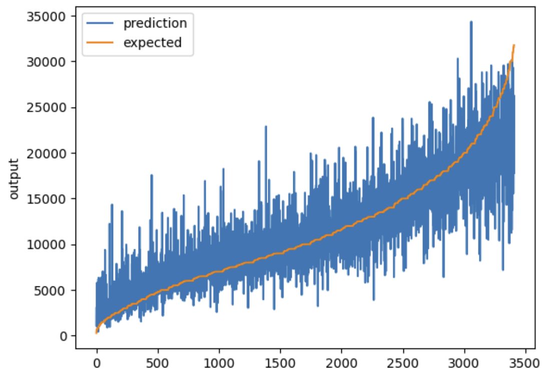


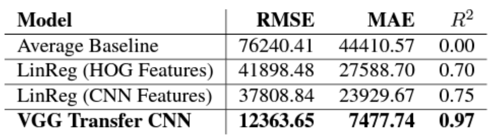
Figure 19: 100-50-25-10 Adam | 4 images VGG Model

4.2.17 Transfer Learning Model - Algorithm B (Figure 19). Transfer Learning Model B had an RMSE score of 3287.807 when one-hot Adv_month and year

5 RELATED WORK

The Price is Right: Predicting Prices with Product Images. Steven Chen, Edward Chou, Richard Yang. Stanford University.

- This group uses two datasets, one for cars and one for bicycles, and estimates the price of the item based on its image.
- Our problem and method mainly focus on a hybrid solution, but their methods include linear regression with HOG, Multi-class SVM Baseline, and similar to ours, they also use VGG16 for transfer learning.
- Our model and problem have the advantage of accessing a much larger dataset, as well as using a hybrid model has been shown to perform better in our tests. Their car dataset consists of 1,400 examples with a price range of $12,000 to $2,000,000. Our dataset has over 250,000 rows of advertisement data, as well as 1,451,784 images. In one of our models, we have 15781 rows of input.
- Thier results (including RMSE): [1]



Model	RMSE	MAE	R^2
Average Baseline	76240.41	44410.57	0.00
LinReg (HOG Features)	41898.48	27588.70	0.70
LinReg (CNN Features)	37808.84	23929.67	0.75
VGG Transfer CNN	**12363.65**	**7477.74**	**0.97**

Figure 20: Results of the other research

6 CONCLUSION

In this project, we have used various model types to predict the prices of used cars. Through this process, we have come to some conclusions about the various model types and parameters used.

Of the 3 categories of models, the CNN models performed the worst, followed by the dense models, and the hybrid models performed the best. For the dense models, we found that adding additional layers (5 hidden layers instead of 4) with higher neuron counts (starting at 200 instead of 100) and using different activation functions (tanh instead of relu for a single layer) had a negligible effect on the model performance. For the CNN models, using all 4 images for each car improved model performance over using a single image. CNNs which used an additional VGG block (2 instead of 1) had improved performance. Using VGG16 with pre-trained weights for the image portion of the model, we found slightly worse performance.

The best performance was found in the hybrid models. Hybrid model B, which used a combination of a higher number of layers and higher neuron count from dense model C, along with 2 VGG blocks for the image portion, performed the best out of all of the models.

7 WORK DIVISION

Everyone helped create skeletons for the models and helped with documentation. Talakoob, Charamuga, and Sullivan did parameter tuning for models. Dinh and Sullivan did data preprocessing and set up the skeleton of the notebooks. Talakoob designed and implemented the four image paths in functional API and experimented with an RNN (LSTM) model which was unsuccessful given the nature of the dataset. Dinh used transfer learning for some parts of the project and created the base for the CNN model. Charamuga designed and implemented the regression models. Sullivan created the base for the Dense model.

8 LEARNING EXPERIENCE

The projects that we worked on during this course were mostly selected by our Professor and were designed in a way that we could easily work on them. For instance, they only needed simple data preprocessing, all the data was in a single dataset with no need for joining tables, as well as having the freedom to limit the size of the database and grab a portion of data instead of all. However, for this project, the team needed to act more independently and choose the topic and database. In this project, since our team worked with a dataset that was over 13.6 gigabytes in size, we learned how to handle larger datasets more effectively. We also had the opportunity to design different models and learn about their advantages over each other as well as the differences in the ways to handle the input and its direct effect on the output. This course also helped us to develop other skills such as teamwork, speech, presentation, time management, and documentation. In fact, this team was formed during the early weeks of this course, and we started working together and communicating with each other since Project 1. This course provided us with knowledge from theory and experience for the projects that we completed. The combination of working on the actual code while learning about the logic and core concepts of it enhanced our learning ability, and yielded a better outcome.

REFERENCES

[1] Steven Chen, Edward Chou, and Richard Yang. 2017. The Price is Right: Predicting Prices with Product Images. *Stanford, cs229 final reports* (2017), 6 pages. https://cs229.stanford.edu/proj2017/final-reports/5237321.pdf

[2] Lan Luo Shigang Yue Jingming Huang, Bowei Chen and Iadh Ounis. 2022. *DVM-CAR: A large-scale automotive dataset for visual marketing research and applications*. Retrieved November 27, 2023 from https://deepvisualmarketing.github.io/