MLPs and CNNs

Saashiv Valjee

saashivvaljee@hotmail.co.uk

Compiled June 12, 2023

Neural networks are a popular choice for solving a wide range of problems, with two commonly used types being the regression MLP and the image classification CNN. A regression MLP is a feed-forward neural network that can be used for regression tasks, while an image classification CNN is a convolutional neural network that is specifically designed for image recognition tasks. Both models are powerful tools for solving complex problems in various domains, and the choice of neural network depends on the problem at hand and the available data.

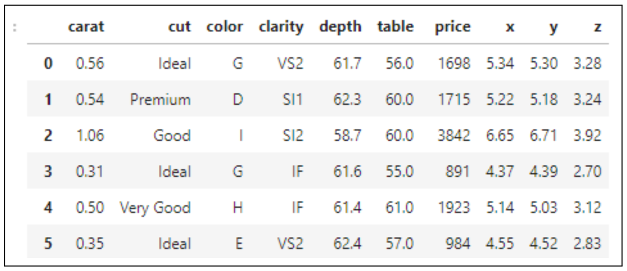
1. INTRODUCTION

In the field of deep learning, neural networks have proven to be powerful tools for solving a wide range of problems. Two commonly used types of neural networks are the regression MLP and the image classification CNN. Regression MLPs can be used to predict the relationship between two variables, and have been used in a variety of applications, including predicting the price of diamonds based on their carat weight. In this report, we will explore the use of MLPs to predict the relationship between the carat and price of diamonds, and the use of CNN's to classify the MNIST data-set into adversarially attacked and non-attacked images. We will discuss the architecture of each model, the preprocessing steps required, and the performance of each model on the given tasks. The results of our experiments will demonstrate the effectiveness of these models in solving their respective problems.

2. DATA HANDLING - MLP

In order to utilize MLPs, we have chosen to use the diamonds data-set (from the pydataset module), which contains a large range of data categories and recordings related to diamonds. Due to the high likelihood of certain categories being correlated with each other, MLP regression models are a suitable choice for predicting the relationship between carat and price. By using MLPs, we can create best fit curves between these two variables, allowing us to gain insights into the relationship between them and make more accurate predictions about the price of diamonds.

After viewing the data we're working with:

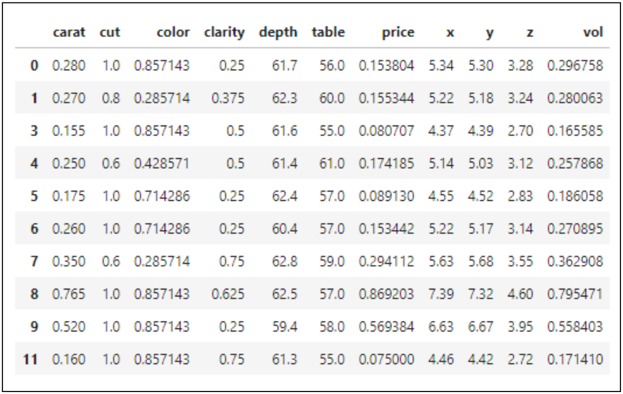


	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.56	Ideal	G	VS2	61.7	56.0	1698	5.34	5.30	3.28
1	0.54	Premium	D	SI1	62.3	60.0	1715	5.22	5.18	3.24
2	1.06	Good	I	SI2	58.7	60.0	3842	6.65	6.71	3.92
3	0.31	Ideal	G	IF	61.6	55.0	891	4.37	4.39	2.70
4	0.50	Very Good	H	IF	61.4	61.0	1923	5.14	5.03	3.12
5	0.35	Ideal	E	VS2	62.4	57.0	984	4.55	4.52	2.83

Fig. 1. first 5 rows of dataframe

It becomes immediately obvious that we will require some preprocessing. The first step is to convert all non numerical entries to numerical entries within the dataframe. Namely, the cut, colour and clarity columns will need converting. This is because Neural networks require numerical data as inputs. Non numerical data cannot be processed by neural networks.

The next step will be to remove outliers. By graphing the data we found anomalous data that had the potential to skew the correlations. To remove the outliers, we categorized each entry based on where in the quartiles of the entire category it lied.
Finally, we normalize the dataframe columns we plan to use column by column. This helps prevent bias in the model due to larger features normally having greater impact on predictions, aswell as helping the convergence speed. The columns I chose to focus on were: carat, cut, color, clarity, price and volume.



	carat	cut	color	clarity	depth	table	price	x	y	z	vol
0	0.280	1.0	0.857143	0.25	61.7	56.0	0.153804	5.34	5.30	3.28	0.296758
1	0.270	0.8	0.285714	0.375	62.3	60.0	0.155344	5.22	5.18	3.24	0.280063
3	0.155	1.0	0.857143	0.5	61.6	55.0	0.080707	4.37	4.39	2.70	0.165585
4	0.250	0.6	0.428571	0.5	61.4	61.0	0.174185	5.14	5.03	3.12	0.257868
5	0.175	1.0	0.714286	0.25	62.4	57.0	0.089130	4.55	4.52	2.83	0.186058
6	0.260	1.0	0.714286	0.25	60.4	57.0	0.153442	5.22	5.17	3.14	0.270895
7	0.350	0.6	0.285714	0.75	62.8	59.0	0.294112	5.63	5.68	3.55	0.362908
8	0.765	1.0	0.857143	0.625	62.5	57.0	0.869203	7.39	7.32	4.60	0.795471
9	0.520	1.0	0.857143	0.25	59.4	58.0	0.569384	6.63	6.67	3.95	0.558403
11	0.160	1.0	0.857143	0.75	61.3	55.0	0.075000	4.46	4.42	2.72	0.171410

Fig. 2. Preprocessed data

After this, we plot the relationships between all the pairs of columns to find candidates for our MLP regression model.

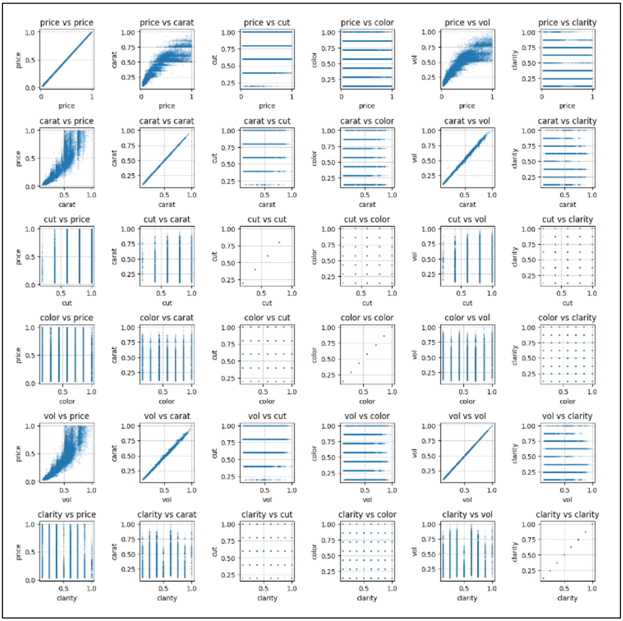


Fig. 3. every relationship between the potential candidate columns

From this, we confirm that carats against price shows some kind of correlation that we can create best fits for, making that pair a suitable candidate.

3. THE NETWORK - MLP

To create the best fit for the relationship between carat and price of diamonds, we will use a simple MLP. MLPs are very well suited for regression based tasks. Regression is a type of supervised learning that involves predicting a continuous target variable based on one or more input variables, here we'll be using price and carat respectively.

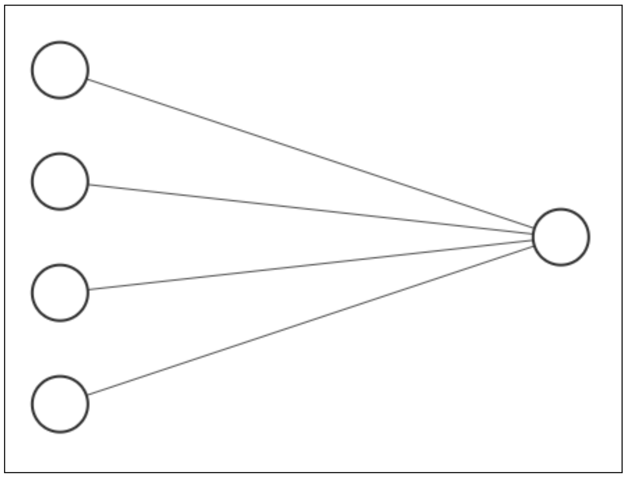


Fig. 4. hidden layer neuron connection diagram. Note that the first layer has 128 neurons, which has been reduced to 4 for display size purposes. Note also that the input layer has been hidden as it's not a hidden layer.

Within the model, the first layer is a dense layer with 128 neurons with the "ReLU" activation function. This is a common choice for regression tasks. ReLU is an activation function that returns the input if it's positive and 0 if it's negative. This behaviour helps introduce non-linearity and prevents the vanishing gradient problem that other activation functions could cause.

Using a 1 neuron layer with the linear activation function is also common for regression tasks. It allows the model to output a continuous value that represents the prediction of the target variable. Using the linear activation model helps the model learn underlying correlations between the data without introducing unnecessary complexity. The simplicity of a single neuron layer with a linear activation function makes it easy to interpret the model's predictions.

To evaluate the models performance, we can plot it's: MSE (loss) , MAE, predictions and the residuals of the fit.

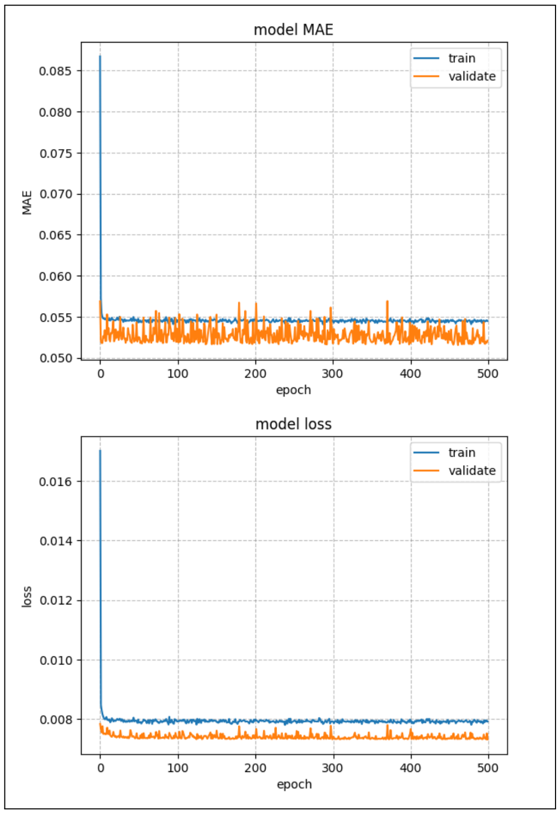


Fig. 5. MAE and loss (MSE)

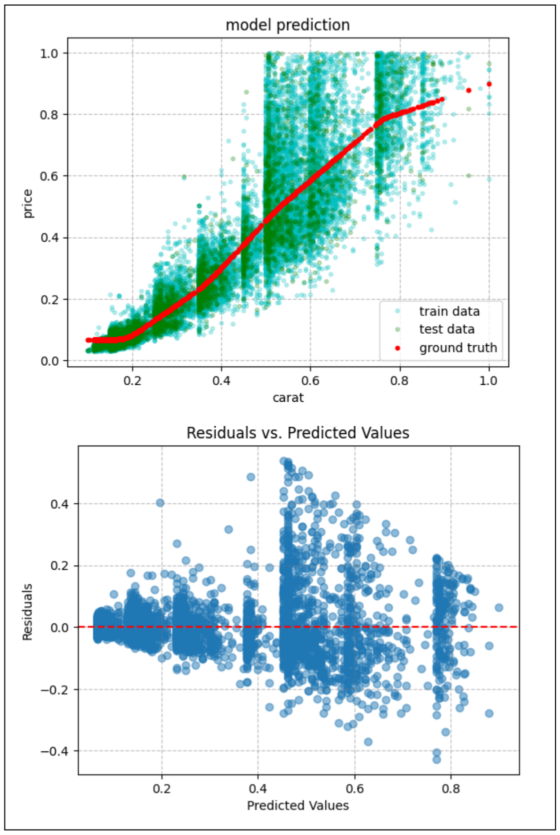


Fig. 6. Predictions and residuals

From our loss and MAE curves, we can say our model has both trained quite well, due to the scale of the differences between the predictions and truth (MAE) as well as the squared difference (MSE/loss). The model has also completed a sufficient amount of training, which the plateaus on each metric suggests. Finally, we rest assured that the model has not over-fitted due to the small difference between the validation and train loss and MAE. The best fit curve in 6 also follows the data nicely. Finally, the residuals have a balanced split above and below 0, further indicating our best fit model has performed well.

4. DATA HANDLING - CNN

CNN's are incredibly proficient at handling and classifying images. With that in mind, our introduced goal is to differentiate between attacked an non attacked images. The first step is to create the attacked images. By loading the MNIST data-set and plotting a fraction of the images, we are presented with the following:

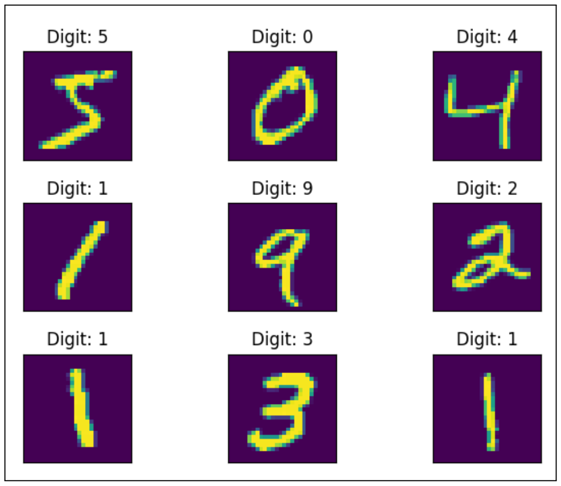


Fig. 7. images before preprocessing

Then, we must create a model to classify these images. To do this, the data structure must represent the dimensionality of the images as well as the amount of images. Having both the train and test data be structured as (amount of images, width, height, color channels) is vital for the model to work. Further more, the data itself must be as floats for the model to create tensor objects from them. Next, the pixel values must be normalized for a more unbiased training and faster training process. Lastly, as we will be using categorical_cross-entropy as our loss function, our labels must be one-hot encoded. This is done using the to_categorical function from keras.

Luckily for us, the MNIST data-set contains images that are incredibly easy to classify, making this a great data-set for our task. We can create a simple and fast CNN to classify the images. After ensuring that model has accurately classified the vast majority of the images, we can create our attacked images. The method for attack will be the fast_gradient_method from cleverhans. The attack on the image itself will be subtle. The goal is not to attack the image beyond recognition from the initial models point of view. The goal is to create a model that can differentiate between images with weak attack signatures.

To do this, we take a random sample of the images from the test data-set and apply our attack function to half of our random sample. After attacking the images, we make sure the format of the attacked image data-set will work with the model as well as the non attacked image set. Next, we can create arrays of labels for the attacked and non attacked images. After that, we combine the attacked and non attacked data-sets, aswell as the labels and shuffle them using the same seed to ensure that the shuffling process is consistent. Finally, we create train and test portions of our attacked and non attacked images and labels for the model.

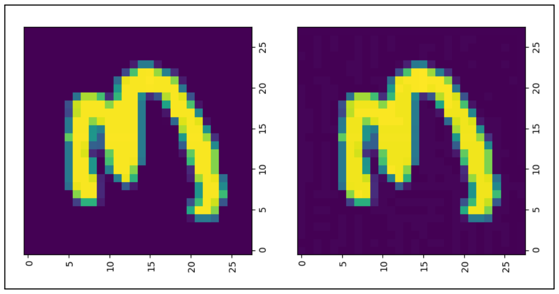


Fig. 8. Example of a non attacked and an attacked image

5. THE NETWORK - CNN

The architecture of the neurons within the network can be visualised as follows:

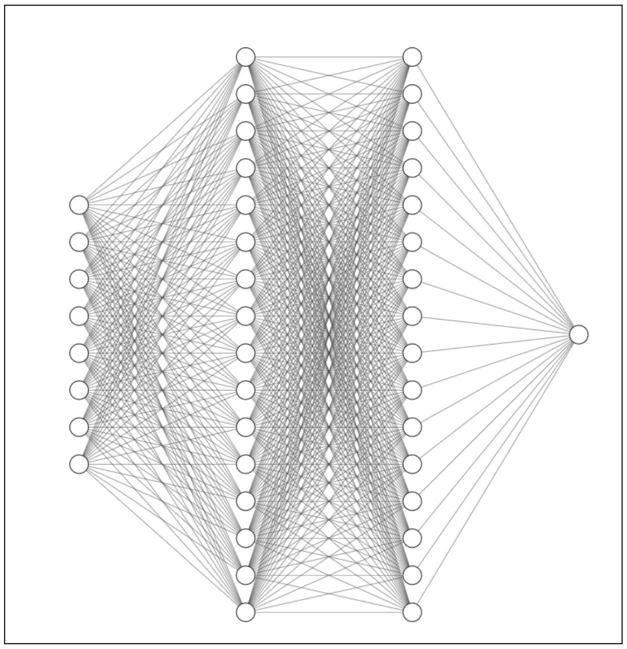


Fig. 9. hidden layer Neuron connection architecture, note that the neuron counts have been scaled by a factor of 4 excluding the final layer, note also that the input layer has been hidden.

The first layer in the network is a Conv2D layer with 32 filters, of size 3x3 each. The activation is ReLU. For our Conv2D layers, we use ReLU because it's computationally efficient with respect to the way it calculates gradients, as well as it's ability to introduce non-linearity to the model which helps CNN's learn more complex features and patters within the data. These layers are responsible for detecting features within the input images.

The second layer is a Max-Pooling layer of pool size 2x2. This layer reduces the size of the feature maps produced by the Conv2D layer. This helps reduce the parameters within the model and therefore helps prevent over-fitting.

The third layer is a Dropout layer of rate 0.25, this layer randomly drops some neurons during training. This helps prevent over-fitting and improves the versatility of the model.

After another set of these 3 layers, we use a Flatten layer. The flatten layer which flattens the 2D feature maps produced by previous layers into 1D vector.

The following layer is a dense with 64 neurons using the ReLU activation function. This layers purpose is to apply a linear transformation to the feature vector and learn to classify the images.

Following layer is a dropout again to prevent over-fitting.

The final layer is a dense layer with a single neuron using the sigmoid activation function. This layer's purpose is to tell us the attack probability the input image has. The sigmoid function is used because it relates to our labels very well. Our labels are 0 and 1 representing non attacked and attacked respectively. The sigmoid function returns our probabilities mapped between 0 and 1. By simply rounding our predictions array, we can obtain prediction labels. This is why the sigmoid function can be favoured when performing binary classification, and why we have chosen to use it here.

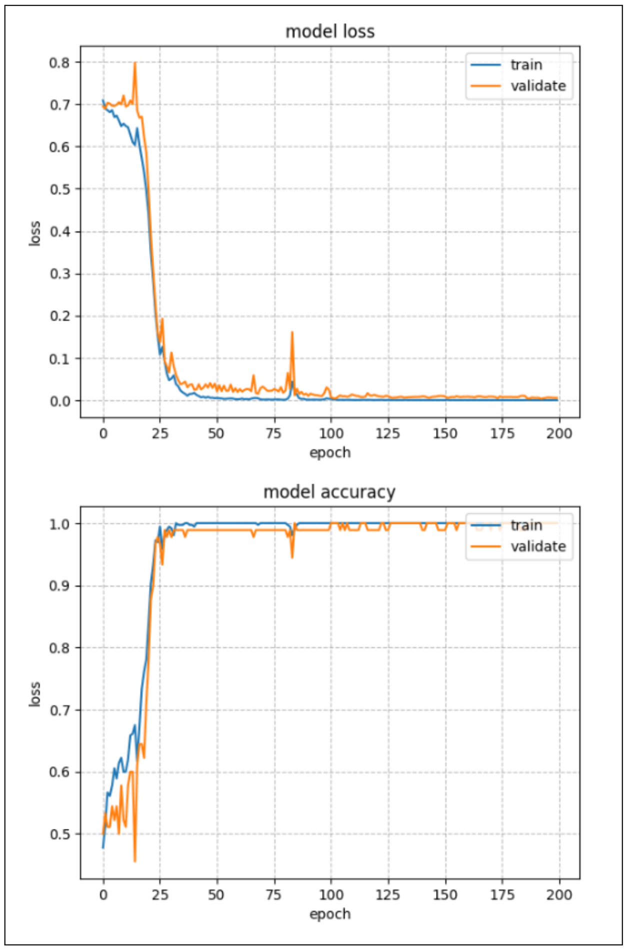


Fig. 10. Loss (binary cross-entropy) and accuracy

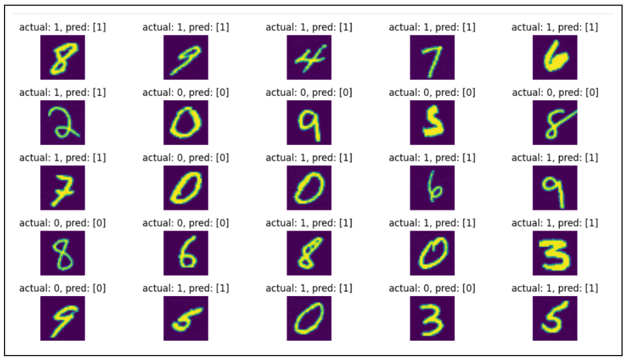


Fig. 11. Plot of some classified images with their actual and predicted labels

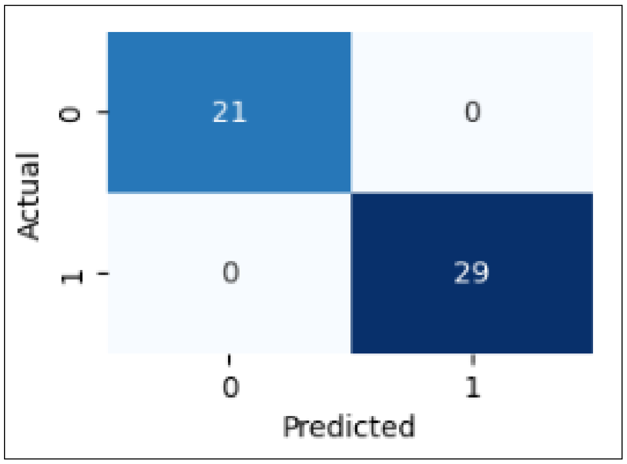


Fig. 12. heat map of correctly and incorrectly classified images

From the loss as well as the accuracy, we know our model is performing well as we see our loss virtually 0 out, as well as our accuracy hitting 100%. We also know our model is not over-fitting because of the small difference between our validation and train metrics in both graphs. Ontop of this, our plot of the images versus their predicted and true titles displays the model performing perfectly, correctly classifying every attacked image. While that is only a portion of the data-set, the heat-map plotting the confusion matrix shows the distribution of true positive, negative and false positive and negative. From the plot, we know that our model has correctly predicted every attacked image within the test data-set.