Vehicle classification with Neural Network tuned by Genetic Algorithm and Knowledge Distillation for Neural Network based on the study of Causal index

Tingwen Yang
College of Engineering and Computer Science Schools,
The Australian National University,
Canberra ACT 2601
u6831034@anu.edu.au

Abstract. Nowadays, Vehicle classification and recognition have become an increasingly critical field because of its great importance in traffic surveillance, traffic congestion prevention, high-way monitoring and etc. Vehicle recognition uses stationary or moving cameras to perform some basic identification task: identify the colour, brand and other properties of a vehicle. This provides detailed and accurate tracking when coping with road accidents and conducting traffic control. Based on the vehicle details provided by a vehicle specification database, vehicle-X, this study proposes a vehicle type recognition method with neural network. Instead of manual tuning, we optimised the hyper parameters with genetic algorithm to reach high accuracy more efficiently. Besides, to understand how neural net works when making decisions, we explored the possible decision making rules that the neural net follows by studying characteristic input and causal index. Also, as it is always difficult and computationally expensive to perform classification task on a complex dataset or with a large neural net , we compressed the model by distilling knowledge from it so that it can be run on smaller devices like mobile phones while validity was kept as much as possible. This would save much computation resources and time cost.

Keywords: vehicle classification, neural network, hyper parameter tuning with genetic algorithm, causal index, characteristic input, neural network compression, knowledge distillation, decision tree.

1 Introduction

In morden intelligent transportation field, vehicle classification and recognition is of great interest and significance, to provide accurate tracking or identification and help achieve high traffic efficiency by minimizing traffic problems. This technique has a wide range of applications: real-time traffic surveillance, incident management, road capacity management, traveller information service, speed control and improvement and etc. These are all closely related to everyone's interest because an advanced morden intelligent transopration system will reduce time of commuters as well as enhances their safety and comfort. Vehicle classification is a basic and important task in the development of current intelligent transportation field, aiming to classify the camera-captured vehicles into different classes.

In traditional study of vehicle classification, scientists used images and information captured by hundreds of traffic surveillance cameras deployed across Canada and the United States to build up the experiment dataset, Miovision traffic camera dataset (MIO-TCD) [1]. However, to collect a large real-world traffic dataset like MIO-TCD is never an easy thing, not only because it is expensive too collect large-scale dataset for muti-camera system, but also an increasing concern over privacy and data security makes the collection less accessible. But to train a classification model with good performance, we need large-scale dataset because in vehicle classification task, one vehicle has many attributes and there is a large variety of each attribute among different cars, like brands, types. And there are even varying viewpoints of a single vehicle, as shown in Figure 1. So it is necessary to collect a large dataset to allow model to learn and identify all these complex scenarios. That's why synthetic daraset is introduced. But some synthetic dataset also has fatal drawback: there is a huge domain gap between synthetic one and real-world one. The gap may be related to object layout, illumination interference, and background in the image. To cope with these, we use Vehicle-X in our study [2].

Fig. 1: Left picture is the 4 viewpoints of the same car. Right is the picture of VehicleID dataset. Each row depicts four view points for two vehicles [3].

Vehicle-X is a large-scale synthetic dataset and is able to generate a large training set by varying object and environment attributes. It scales up the real-world training data without concerns of labelling issue or privacy, and at the same time reduce the content domain difference with real-world data. VehicleX ideally approximates the attributes in real-world datasets and provides adequate and robust vehicle attribute information to train a vehicle classificatio model. Specifically, created in Unity, this dataset contains 11 types vehicles of various 3D models and provides a training set of 45438 images, validation and testing set with 15142 and 14936 respectively. These images are extracted from a pre-trained ResNet so that the model trained on this dataset can classify the real-world vehicles based on their types (SUV, Sportscar and etc), which is similar to the idea of transfer learning.

To conduct this type classification task, we implemented traditional neural networks with different structures. Also, we employed Genetic Algorithm to efficiently auto-tune the hyper parameters and optimise the neural net model. Additionally, as this is a complex classification task, the decision making process of the neural net is just like a “black box”, to figure out how the model might work as much as possible, we studied the characteristic input and causal index. Characteristic input is a statistical representation of those inputs that generated the same output and it helps with the rule extraction and increases the computation efficiency when using causal index to explain the work that a neural net has done. And we found out that almost every input feature is equally weighted in the deicison making rules of the NN. As the encoded features are not quite worthwhile to interpret because they are all synthetic without too many actual meanings, we used the finding of causal index study, conclusion drawn from PCA feature extraction and Decision tree model to compress the original model (reduce input feature number and simplify its archietecture) rather than build up an explanation mechanism. We compressed the model using knowledge distillation technique. This enabled our model to run on smaller hardware devices like mobile phones and save much computation resources and time.

2 Method

The hardware platform of this paper is: Operating system: macOS 10.15.7 10; Memory: 8 GB; Processor: 1.4 GHz Quad-Core Intel Core i5; Software: python3

2.1 Neural Network

To perform the main task of classification, we implemented neural networks of one and two hidden layers respectively. These two models are traditional nulti-layered neural net (also known as Multi-layer perceptron, MLP). We have 2048 features and 11 classes to classify, so we set the neuron number in input layer as 2048, and that of output layer to be 11 for all NNs. There is a link associated with a weight between every neurons from two neighbouring layers. For each hidden layer and output layer, there are normally two related functions: one is a linear function taking the input passed to this layer, the other is an activation function like Softmax or ReLU(only used in hidden layers) to decide how the weighted sum of the input is transformed into an output from a node or nodes in a layer of the network. These activation functions performing non-linear operation are necessary and increase the robustness and fitness of NN, because a NN without activation functions is just a linear regression model. These traditional neural net models employ the supervised learning technique known as training backpropagation. MLP differentiates from a linear perceptron by the numerous layers and nonlinear activation. It can differentiate between data that cannot be linearly separated. When first implemented baseline NN models, the hyper parameters are tuned roughly using validation set. And we set other hyper paramters of these two model to be the same to better show the impact of different archietecture. Below are the accuracy of NN with different archietecture after 1450 and 1500 training epochs:

NN structure	Hidden layer neurons numbers	Accuracy after 1450 epochs	Accuracy after 1500 epochs
One hidden layer	100	44.76%	44.99%
Two hidden layers	500, 100	44.92%	42.18 %

The accuracy result indicates neural network with 1 hidden input (simple 2 layered NN) works better on vehicle-X dataset. The fact is that hidden layers are used for better classifying some boundary cases, that is suppose we are classifying a convex or concave region with arbitrary complexity region in a 2D plane[5], which illustrates the use and advantage of hidden layer. However, increasing number of hidden layers also means longer computation and training time and may even lead to overfitting problem because in this case we have large amounts of training samples but only have 11 classes which means if there is any bias on either of the classes, the difference would be amplified. So in this case, NN with fewer hidden layers works better.

Also during the implementation of the baseline models, we rougly and manually tuned the hyper parameters using validation set. A more accurate and effective tuning method will be introduced later. So below are the varying accuracy of the two-layered model with different hyper parameters:

Number of hidden layer neurons	accuracy
5	23.00 %
100 (chosen one)	41.80 %
500	42.74 %

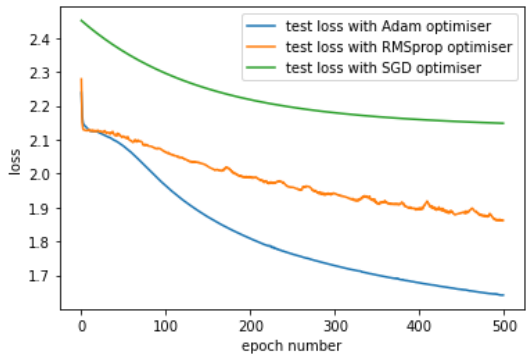


Fig .2 accuracy of models with different hidden layer neuron numbers

optimiser	accuracy
ADAM (chosen one)	41.49 %
SGD	21.21 %
ASGD	32.20 %

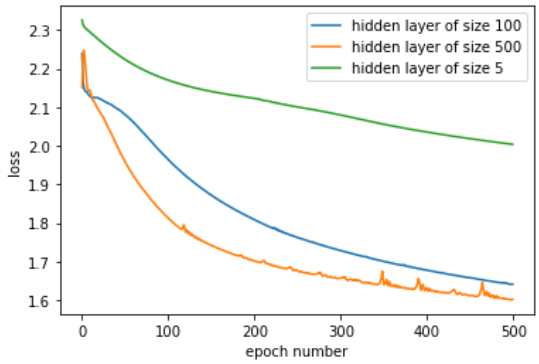


Fig .3: accuracy of models with different optimisers

During the whole tuning process, we just used the validation set to tune the parameters rather than apply K-cross-folder validation methods. Because the ratio scale between the train, validation and test (45438:15142:14936) is roughly 6:2:2, and that indicates the size of our given validation set is large enough to ensure almost every observation from the original dataset has the chance of appearing in the experiment.

For the activation functions, we used Softmax for output layer and Sigmoid for hidden layers. Because output layer may need a probability-distributed output prediction to better reduce the loss. And for other layers, Sigmoid can provide smooth gradients, preventing sudden and relative huge change in output values. Also Sigmoid enables clear predictions because for input value out of the certain range, Sigmoid will bring the corresponding output value to the edge of the curve, close to 0 or 1. Next, we chose the hidden layer size of 100 for two-layered NN and 500, 100 respectively for four-layered NN with 500. But from figure 2, the models with larger hidden layer performs better, but we made a trade off between the training time and the accuracy. The the two models don't differ a lot in accuracy but the training time of the larger NN is as two times much as that of the smaller one. Finally, the application of ADAM optimiser together with the learning rate largely increases the accuracy of my NN from around 20% to 40%. As ADAM combined bias correction and momentum based on RMSprop, and when the gradient gets sparser, it has better effect than RMSprop, and normally ADAM is not easily stuck in a local minimum as SGD does [10]. Smaller learning rate will increase the possibility of landing at the optimal minimum and thus possibly helps reach a more precise result.

Above all, we finally choose traditional two-layered NN with 2048 input neurons, 100 hidden neurons, 11 output neurons which is the number of type classes, together with learning rate of 0.001 and Adam optimiser and three-layered one with input layer of 2048 size, hidden layer1 with 500 neurons and second hidden layer with 100 layer, optimiser Adam with learning rate 0.0001. This is just a rough hyper parameter tuning process, we will introduce a more effective one next.

2.2 GA for hyper parameter tuning

Hyper parameter tuning is quite critical for a neural network performance and indeed the choice of optimal parameter enables the accuracy of the NN models to improve a lot. As we can see from the figure 3, there is a huge difference between the effects of varying optimisers. So, parameters need to be tuned to construct the optimal classification model. However, the search space of potential combinations of parameters options for these classification models are almost infinite and endless. So, a heuristic tuning method is expected to deal with this case. That's why we use GA for hyper paramters optimisation rather than manually adjust those paramters.
Genetic algorithm provides us with an effective and powerful way to tune hyper parameters of a machine learning model or a neural network model. Generally, genetic algorithm is implemented based on the concepts from evolution by natural selection. GA uses Darwin's "evolution theory" and "survival of the fittest"to find the optimal solution to a given problem. By using GA, the exploration of large search spaces can be conducted in a time efficient manner, making it an ideal approach for problems like feature selections, parameter optimisation and etc. Random parameter mutations are introduced in the population, and vectors with a larger fitness outlive other vectors. The overall procedure of GA is shown in figure 4:

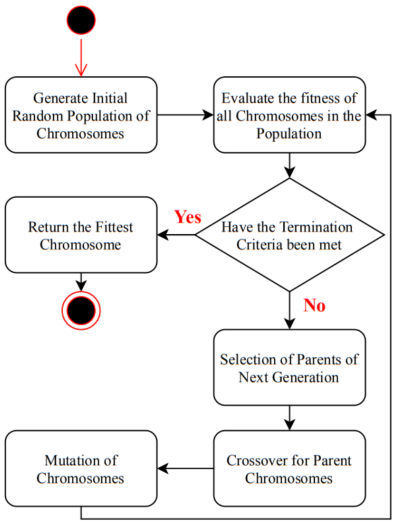


Fig .4 flow chart of GA

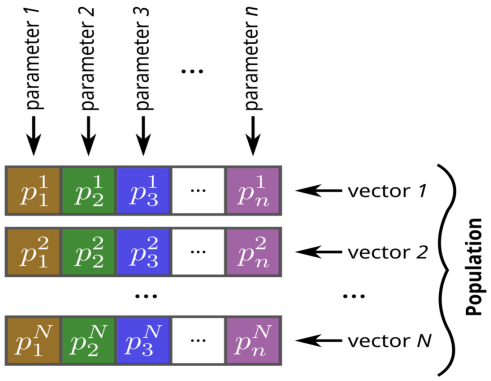


Fig .5 illustration of the role of parameters, and Combinations of different paramters' values.

The process of how GA envolves an optimal solution for hyper parameters tuning problem can be devided into several steps, and we used python library DEAP (provides novel evolutionary computation framework) as main tool to implement GA:

- Generate original population:
  Parameters correspond to genes in biological system. We encoded the parameter values into binary system. The values of different parameters compose a vector, and each vector is considered to be an individual(chromosome). Each individual represent one candidate solution. So the population under this scenario is the combinations of different paramters' values. Figure 5 illustrates the idea. For our model (two-layered model), each chromosome is a binary array and has the length of 19 and represents the values of two hyper parameters: hidden layer number and learning rate. The first part of length 9 is related to the value of hidden layer number while the left part of length 10 is related to learning rate. We represent the values of these two paramters following this formula:
  Let us use b1 to denote the binary number extracted from the first part of length 9 in the chromosome
  b2 to denote the binary number extracted from the latter part of length 10 in the chromosome
  hidden layer number = 100 + b1 value in decimal system
  learning rate = $\frac{1}{100+\text{b2 value in decimal system}}$
  So we generate the initial population by using toolbox.binary from DEAP and random binary arrays are generated as the indivisuals composing the initial population.

- Evaluate individual(chromosome) fitness:
  Each chromosome is assigned a fitness score by a fitness function and this score reflects how good a solution is. In our research, we used the testing accuracy of the neural net model with the chosen hidden layer number and learning rate as the fitness function. So the objective is to maximise the fitness value, so we set the weights variable in creator.create('FitnessMax', base.Fitness, weights) to a tuple start with 1.0. If we are looking for the minimum mean square error, then we set the first entry in the tuple to be -1.0.

- Select parents for next generation:
  After each individual has been evaluated and assigned with corresponding fitness score, we selected the fittest individuals and used them as the parents for generating the next generation. And those haven't be chosen were removed. There are many implemented operators for selection, and the most common one are selTournament() and selRoulette(). In our research, we chose tournament selection technique with size 3.

- Crossover for parents and possible mutation:
  After the parent chromosomes are selected, they need to evolve the next generation (offsprings). Evolution consiste of two parts: cross over and mutation. Crossover, also known as recombination, determines which solutions are to be preserved and allowed to reproduce and which ones deserve to die out. Different from reproduction, recombination is based on "Selects The Best, Discards The Rest". This principle is accordance with the ultimate objective of our GA， so we hoped to find a way to combine two fit individuals will produce an even fitter individual solution. Again, there are many ways to perform chromosome crossover and for our research, we chose tools.cxOnePoint(). This crossover operator executes a one point crossover on the input sequence individuals. The two individuals are modified in place. The resulting individuals will respectively have the length of the other [4]. Besides, crossover probability is the chance that two chromosomes exchange

some of their parts and the crossover probability was set to 0.6 in our research.The illustration of this crossover is shown as below:

Chromosome1	11011|00100110110
Chromosome2	11011|11000011110
Offspring1	11011|11000011110
Offspring2	11011|00100110110

Single Point Crossover

Fig 6. An example of one point crossover

Mutation is the process during which a string is deliberately changed so as to maintain diversity in the population set. And this change happens randomly with certain probability and some value in the chromosome is altered to a random value. Mutation normally occurs with a very low probability and is used to reduce the possibility of being stuck in local optima (maximum, in our case). In our research, we use the mutation operator: tools.mutShuffleIndexes(). And the mutation probability was set to 0.1.

- Iterate the process until the optimum is reached or the optimization process is not progressing:
  The steps of fitness evaluation, selection, crossover and mutation are conducted iteratively until some termination criteria is met. A termination criteria might be the achievement of a certain fitness function value, or optimization process is not progressing, and also be the case when a maximum number of generations of evolution. When GA terminated, we obtained the optimal solution by using tools.selBest(). There is one parameter "k" in this function, which enables us to get the top k solutions (individuals) from the last population of the whole evolving process.

2.3 Characteristic input and Causal index

We used the test set to calculate the characteristic inputs of different pattern and different pattern refers to varying vehicle types. Those characteristic inputs will be used for calculating causal index, which would be more efficient than using the entire testing set for calculation. To compute the characteristic inputs, first the test data set was fit into the 2 layered NN which has been previously trained. Then based on the output generated from this neural network, we grouped the inputs data based their corresponding output predicted by NN. In our experiment, the output type values predicted by NN consist of 10 kinds of different values. So the inputs (test dataset) was divided into 10 groups. Each of these 10 groups is corresponding to a characteristic Type_α group, where α represents the type value of a vehicle. So the mean value of each group of data is the the characteristic input of that group.

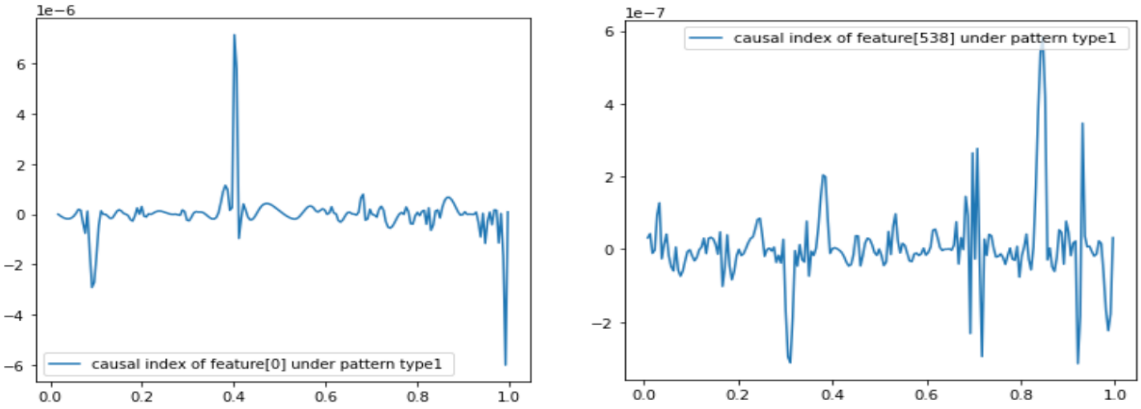
Then, we conducted classification task based on those characteristic inputs: the result of the prediction (classification) is the class label of one of those 10 characteristic inputs. That class label belongs to one characteristic input that has the smallest distance or difference from the given input. We evaluated the accuracy of this prediction, which is only 15.71 %. The accuracy of the prediction based on characteristic inputs is much smaller than the accuracy of the original two layered model (44.99%). So this indicates that NN sometimes makes unexpected decision that some inputs is classified as a specific class even though that input does not have common attributes of the most samples labelled with this class. Because characteristic inputs represent an average and general situation, so we can tell some inputs are still categorised into one class though its features is different from the typical features of this class. Following table indicated characteristic inputs for characteristic Type_0 group and characteristic Type_10 group respectively. As we have 2048 input values in a characteristic input, so only part of them are listed.

Table 1. The characteristic inputs for different groups, only shows those for pattern with type 0 and 10.

Characteristic Input for pattern type 0	…	Characteristic Input for pattern type 10
[0]: 0.489074		[0]: 0.520129
[1]: 0.435392		[1]: 0.365705
…		…
[2046]: 0.471104		[2046]: 0.401623
[2047]: 0.412550		[2047]: 0.362211

Next, after getting characteristic inputs, we calculated the causal index based on the formula provided in this paper[7]. Generally, causal index is the rate of change, i.e. gradient of the output with respect to the corresponding input. Because causal index is discussed based on the characteristic input groups, so for each input (feature), we created new datasets and there are in total 1000 sample data in each of those datasets, and only the feature whose causal index needed to be calculated is assigned with a random normalized value in its original range value, which is [0,1], and other features are assigned with the corresponding characteristic values for that feature [8]. And these newly created datasets are fed into the 2 layered neural network which has been trained previously with original input dataset. By using *tensor.backward()* and the *grad* attribute of tensor in torch library, we got the gradient of output with respect to the corresponding input. And we plotted the rate of change (gradient) of the output as input changes within in range [0,1] as below. As there are too many features in our data set, so the following are just the images of causal index of feature indexed 0 and indexed with 538 and they are both in characteristic Type_1 group.

Fig. 7 Causal index of different feature (input) for type 1 group. The left one is the image of causal index of feature with index 0 for pattern type 1 and the right is the image of causal index of feature with index 538 for pattern type 0. X-aixs represents the input value and y-axis stands for its corresponding gradient value.



Based on the principle of extracting rules from the causal index in this paper[6], according to the images above about the relationship between input value and its gradient. When the causal index values reach a positive or negative peak and then change to 0, the value of corresponding input at this time is considered as the important clues for rule extraction because this input value is the boundary value in the rule. We recorded these points where causal index reaches 0 and tends to be gentle, i.e. the rate of change of output w.r.t reaches 0 and tends to be gentle. These points indicate that at this time the output reaches a local maximum or local minimum, which can affect the decision making of the neural network to some extent. For instance, in the left image of figure 7, we can find that the point where gradient flips is around 0.17 and 0.44 along x-axis. And as this image is about type 1 (type 1 pattern). So "X[0] < 0.17" or "X[0] < 0.44" will be add in the rules for characteristic type1 pattern. That's how a rule of one feature related to one pattern (type_alpha) is extracted. For different characteristic groups, there are 2048 features within them, and each of the feature would be possibly related to an important point in its causal index image like above, so we can draw more than 1 rules generally. All characteristic groups are consisted of sub-rules like this. Then after finding all the rules for a specific characteristic type alpha group, those rules are connected with "join" operator, because in this case as we have more than 2 classes, so the original ON and OFF groups mentioned in [6] are extended to many "ON groups", because for instance if we think the group for type 0 as ON group, then it is actually the OFF group for all the other type_alpha groups.

Referring to this article[9], in order to check whether the extracted rules are good enough to describe the NN, we found all the features mentioned in the rules, and ignored those are not in the rules. Then we put all those mentioned features together as a new input dataset and re-train the NN model. Then we compared the result with the accuracy of the original NN with all input features. If the variance between two accuracy is small, that indicates a good rule extraction. The detailed result of this evaluation will be discussed in section 3.

2.4 Knowledge distillation

In recent study of various classification tasks, more and more complex and larger models have been introduced to cope with some complex classifaction problems. These models are huge and have demanding computation requirement. There is no doubt some complex models perform quite well, however, they consume many computation resources and are trained with expensive cost. What's more, they cannot be run on smaller devices, that is some edge devices like smart phone. So to save computation resources and extend the application of those models, the idea of Knowledge distillation is introduced. Knowledge distillation, also known as KD. KD is broadly applied in the field of knowledge transfer and model compression[9]. Also, KD is always summuraised and characterised by "Teacher-student" learning pattern. KD is proposed to transfer information or the obtained experience in one model (usually a more complex or larger one) to another (normally a small model of much simpler architecture).

When we use KD for model compression, the process is different from network pruning and network quantization. We use a more complex model A to teach a simpler model B. With the help from A, B will exceed itself and learn better. Then we can deploy model B to some edge devices like smart phones. The main intuition (the core idea) of KD is that: the teacher net will transfer some "dark knowledge" to its student, and that is related to the radical objective of machine learning: after training, obtain the model with great generalisation on the given classification task. Then what is the "dark knowledge" and how is that related to the model generalisation ability. We will use figure 8 to illustrate the idea. Before starting, let's compare the traditional training process with the KD training process:

- Traditional training (hard targets): compute maximum likelihood based on the ground truth
- KD training process (soft target): use the class probabilities in the large model as soft targets

Fig .8

So for the ouput of softmax layer, except for positive example (probability of real class), the negative labels (the probability of other classes) also include large amounts of information, for instance, the probability of one negative class is much larger than the probabilities of the other negative classes. This large amounts of information is the "dark knowledge." In figure 8, from left to right, we see 3 kinds of fruits: tomatoes, persimmons and bananas. In a traditional training process, we get the ouput [1,0,0] for tomatoes, and we draw the conclusion: this is an tomato. While the teacher net in KD will output: [1,0,0] + [0.7,0,29,0.01]. This means though we classify this object as tomato and the latter two labels are both negative, we also extract an inclusive information: a persimmon looks similar to a tomato to some extent. This is the dark knowledge in this example. We can see there is a difference between the probabilities of negative class labels and this information is related to the generalisation ability of a model. In traditional training process, all the negative class labels are equally treated. But things are different in KD. The information about negative labels are also extracted and used. That is why we say the teacher net brings more knowledge to the student net, compared with traditional training. But how can we extract the dark knowledge and transfer it to student net? That is how to quantify the difference between "a tomato" and "a persimmon"? The "low temperature" models (those we traiditonally implement without temperature in softmax) perform well at hard prediction but lose the dark knowledge. But a "high temperature model" will have the dark knowledge. The definition of temperarure will be explained later. So the main idea behind KD is transferring this dark knowledge from a well trained teacher to a simpler student model.

Then we will introduce the detailed overall process of KD, below image clearly illustrates the whole idea:

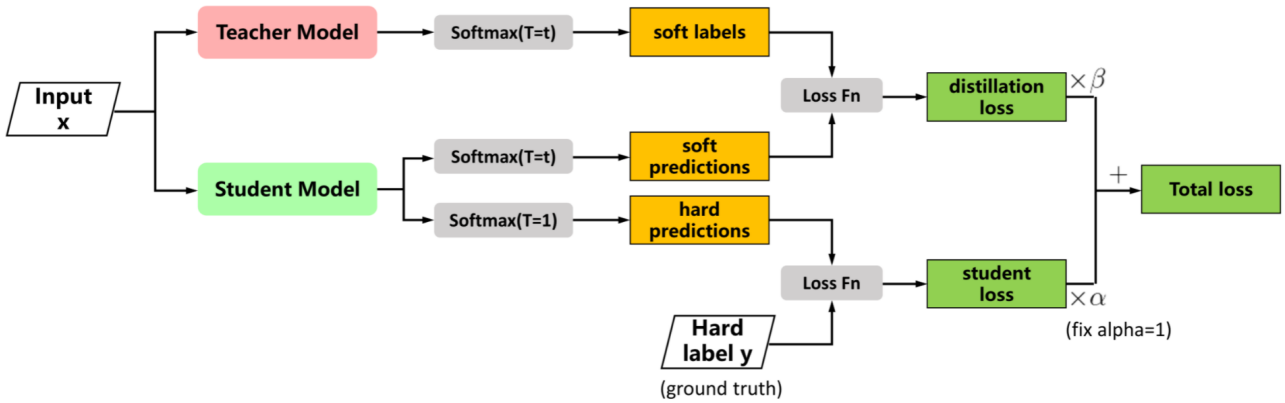


Fig .9 KD process

The loss function in distillation process is as following:

$$L = \alpha L_{soft} + \beta L_{hard}$$

where

$$L_{hard} = -\sum_{j}^{N} c_j \log(q_j^1) \quad \text{Where } q_i{}^1 = \frac{\exp(z_i)}{\sum_k^N \exp(z_k)}$$

$$L_{soft} = -\sum_{j}^{N} p_j^T \log(q_j^T) \quad \text{Where } p_i{}^T = \frac{\exp(v_i/T)}{\sum_k^N \exp(v_k/T)}, \; q_i{}^T = \frac{\exp(z_i/T)}{\sum_k^N \exp(z_k/T)}$$

In the loss function, hard loss is the student net loss and soft loss is the distillation loss. As we know, the original softmax function is: $\frac{\exp(z_i)}{\sum_j^N \exp(z_j)}$. If we directly use the output of softmax layer as soft target, when the entropy of probability distribution is relatively small, possibilities of all the negative class labels are close to 0, this will contributes little to the loss function. So we introduce "temperature " variable (T) and add that into the softmax function, then we have: $\frac{\exp(z_i/T)}{\sum_k^N \exp(z_k/T)}$ as shown above in the loss function of soft target. Original softmax function is a special case when T=1. When temperature T is larger, then the distribution (curve) of the output probability from softmax layer will be more smooth, the entropy of the distribution will be larger, then the dark knowledge stored in negative class labels will be relatively emphaised more. Then the model will pay more attention to the negative lables during training process. The whole KD process can be divided into two stages: 1) train techer net (net-T). 2) under the temperature T, distill knowledge from net-T and train student net (net-S). There is no limitation for net-T about archietecture, parameter number, integrated or not, the only restriction is that the for each input x, there is a corresponding output y which can be mapped to a probability of a class via softmax. The first stage of KD is similar to any other traditional training process. In the second stage, the loss function we used is the one shown above (loss function in distillation process). Net-T and net-S both take the training set used for training net-T in first stage as the input. Then use the softmax distribution (with high temperature) generated from net-T as the soft target, the cross entropy between softmax output of net-S under the same temperature T and the soft target is the first part in distillation loss function, L_{soft}. The cross entropy between softmax output with T=1 and the ground truth label is the second part of distillation loss function, L_{hard}. The reason why we need second part L_{hard} is that net-T is likely to make mistake, using ground truth label will effectively lessen the possibility of transferring the mistake to net-S. That is the student not only learns from teacher, but also learns (refers to) standard answers.

3 Experiment results and Discussion

In this section, we will discuss the experiments we have conducted and interpret the possible hints we have got during the implementation process of the models and those above techniques. For the first part we will demonstrate our exploration on the dataset and some pre-processing work. Then, we will illustrate the evaluation of our models and visualization results of the implementation of those techniques. Finally, we will compare the different performances of the models and show how the techniques improve the model, and also we will discuss about the possible reasons and some thoughts of all those results.

3.1 Dataset inspection and Data pre-processing

We used the synthesis dataset is generated by VehicleX (the publicly aviablable 3D engine). There is in total of 1362 vehicles of 11 different types like SUV, sportscar, van and etc. And there are respectively 45438, 14936, 15142 images in training, validation and testing datasets. Our task is to classify the vehicles based on their types. We plot 3 piecharts and one barplot to illustrate the distribution of different types in train, validation and test sets as below figure 10. We can see that the overall distribution of types in 3 sets are quite similar. And within each dataset, the 11 types are not evenly distributed. We can see in first type (sedan), fourth type (hatchback) and the last second type (sportscar) appear more frequently. Sedan and hatchback types both takes up around 20% of each dataset and the sportscars take up around 17%.

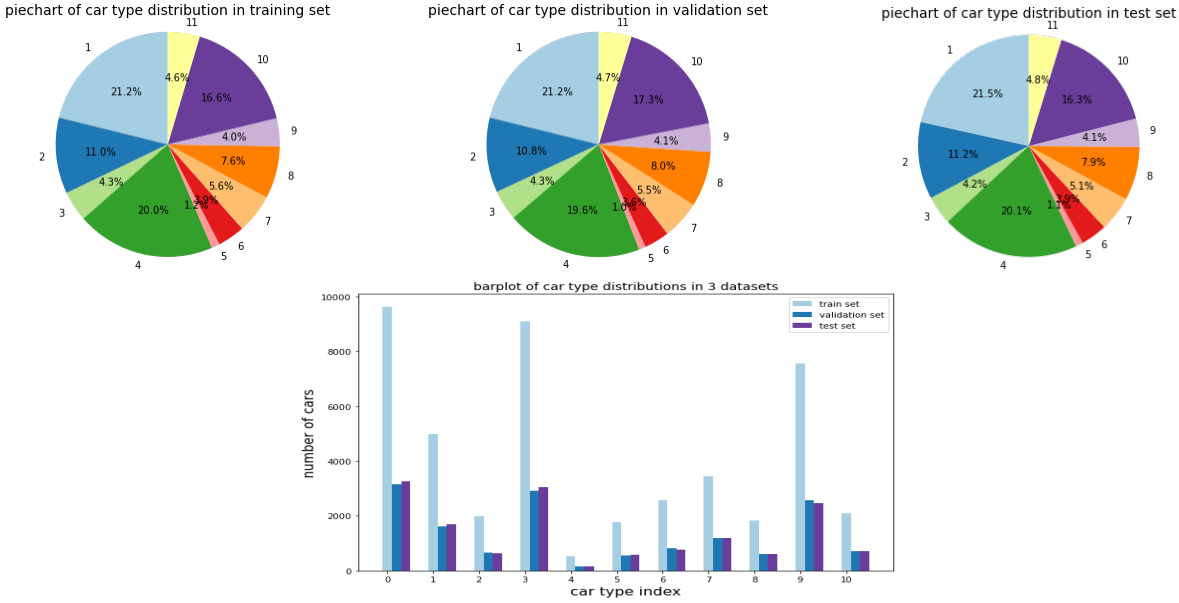


Fig .10 type distribution in train, validation and test set

Each data sample consists of 2048 image features extracted from ResNet which was pre-trained on ImageNet[2]. And each sample is a single feature vector stored in a ".npy" file named "id_cam_num.npy". For example, in "0001_c001_33.npy", 0001 means the vehicle id is 0001, c001 means the camera id is 001 and 33 is the counting number. Also, detailed labelling include vehicle orientation, light intensity, vehicle type and etc is provided in an "finegrained_label.xml" file. The target (ground truth label) in our classification problem is the vehicle type, indexed with number ranging from 0 to 10, representing Sedan, SUV, van and etc. So we summarised the feature information stored in separate ".npy" files and then joined the information with the types labels stored in ".xml" file on their common attributes: the ".npy" file's name. We fed the feature vectors of length 2048 into the model as the input and the type labels as our ground truth target. It is good that 3D image engine generates enough synthesis features for us to conduct vehicle classification task.

Also, to make the model more accurate, we did some pre-processing work on the input data. As the value ranges of different features are not uniformed. We take feature with index 1107 as an example. We plot the frequency histogram of this feature before being pre-processed and after then. We can see in the left histogram of figure 11, there are many values larger than 1, for instance, there are 4279 values of feature 1107 in range from 1.05 to 1.14. So we normalise the data and map every value of each feature to range [0,1] and at the same time keep the original distribution. So we apply this formula for normalisation:

$$x_{new} = \frac{x_{old} - \min(X)}{\max(X) - \min(X)}$$

where X is a single feature vector and x_{old} is the old value of each entry while x_{new} is the processed entry value. This normalisation work will reduce the interference of the different features' value ranges. Because if two features have different value ranges, one is quite small and the other one is quite large. The larger value range will intrinsically impact the training of our model because different ranges of values will cause the case that the gradients may oscillate back and forth and take a long time before it can finally find its way to the global optimal. And for instance, there are two features f1 and f2, f1 ranges from to 0 to 0.5 and f2 is from 0 to 1000. A change of f1 of 0.5 is 100% change while a change of f2 by 0.5 is only a change of 0.05%. This will disturb the training of neural nets. Also, for the feature with larger range, it is large but it doesn't means it is more significant as a predictor. So normalization copes with these problems well by unifying the range (scale) of different feature values and also it maintains the general distribution and ratios in the source data. As we can see in figure, the distributions of left and right are almost the same, and they both follow a bell-shaped normal distribution.

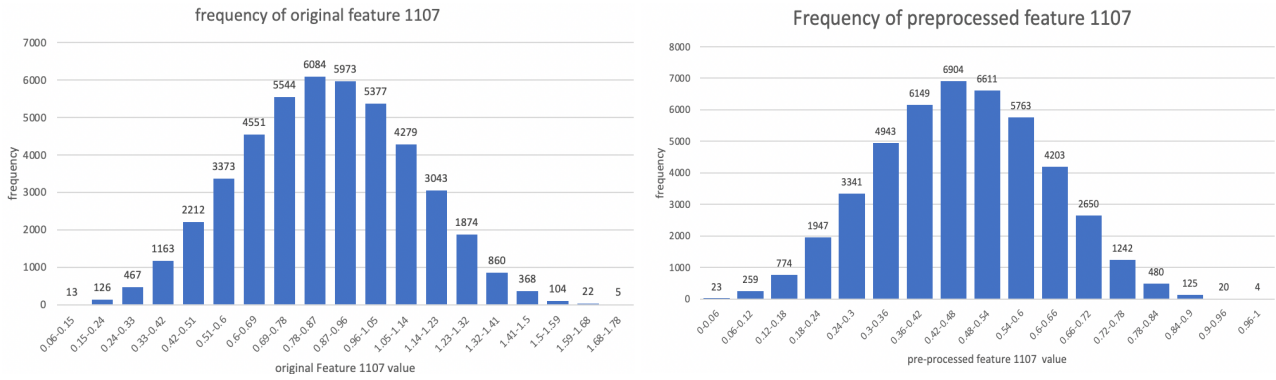


Fig .11 frequency histogram of original features and pre-processed ones

3.2 Neural network performance with tuned hyper parameter by GA

We have conducted several experiments on exploring GA's performance when choosing different operators. We adjusted the generation number, selection operator and crossover operator. The different performance of GA is evaluated by the accuracy of NN with its final optimal solution for hyper parameters setting. And the detailed difference in performance of GA is shown in table 4:

Table.4

Generation number	Optimal hidden neuron number	Optimal learning rate	Accuracy of optimal model
2	275	0.0016260162601626016	45.93 %
6	310	0.0013280212483399733	46.05 %

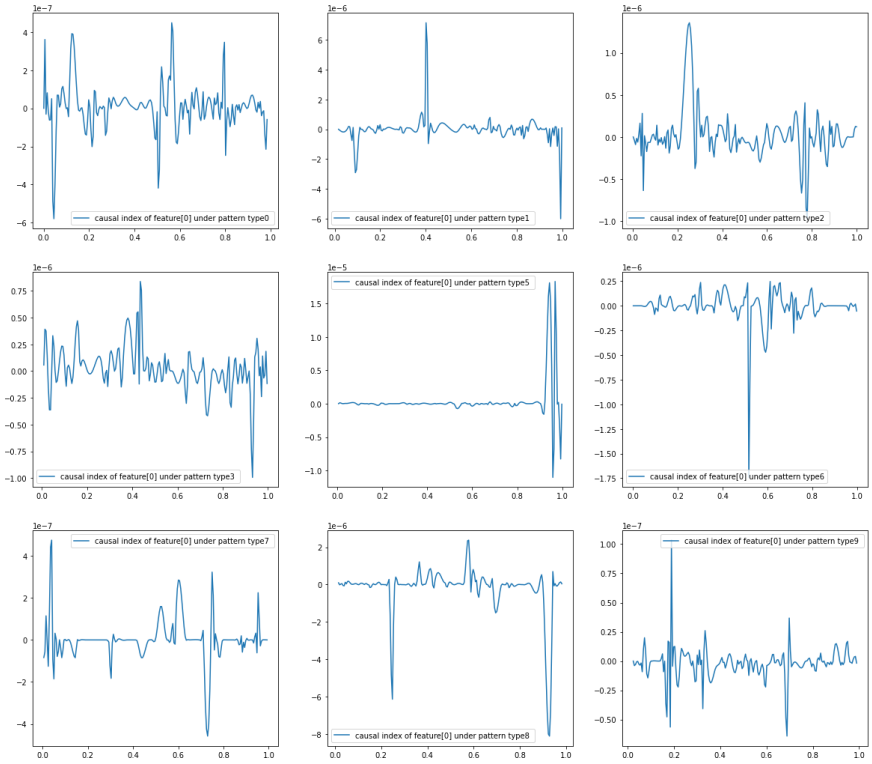
Selection method	Optimal hidden neuron number	Optimal learning rate	Accuracy of optimal model
selTournament with size 3	310	0.0013280212483399733	46.05 %
selRoulette	561	0.0012987012987012987	44.82 %

Crossover method	Optimal hidden neuron number	Optimal learning rate	Accuracy of optimal model
cxOnePoint	310	0.0013280212483399733	46.05 %
cxTwoPoint	388	0.002583979328165375	44.07 %

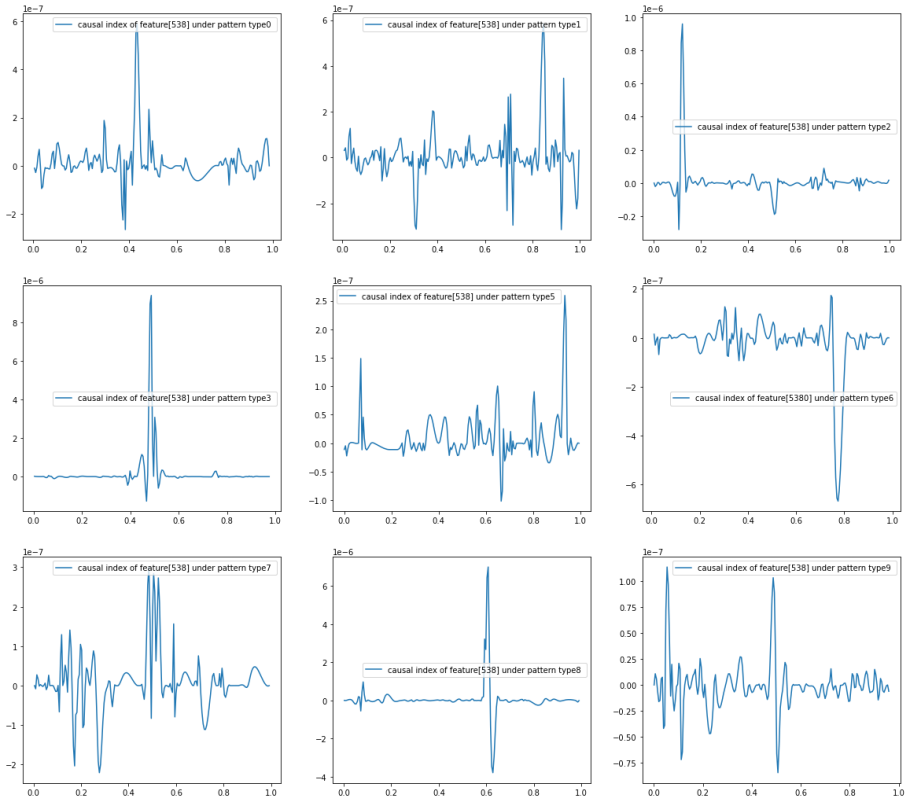
Except for those parameters, we set initial population size to be 3, mutation operator to be tools.mutShuffleIndexes, fitness function to be the accuracy of NN, crossover probability of 0.6 and mutation probability of 0.1. From the above table, we can tell GA with larger generation will perform better than with smaller generation. Because larger generation will enlarge the search space and is more likely to include the optimal solution. Besides, tournament (with size 3) selection method worked better for our experiment, which is normally the case in most practical researches, e.g. the research conducted in this paper [12]. The tournament selection is to randomly select few individuals from the population to participate in the tournament and then the individual with the best fitness (winner) is selected for crossover. And the roulette wheel selection will first compute the sum (S) of all fitnesses in the population, then it generate a random number x in the range [0,S], finallt it goes through the population and sum fitnesses. When the sum s is large than x, it terminates and return the current individual. So the probability of being selected is proportional to its fitness score. So compared to roulette wheel selection, tournament selection normally converges more quickly in practice. Additionally, we noticed that the most simple crossover method “cxOnepoint” outperform than the “cxTwopoint”. That is might because when using single-point crossover, offsprings will be less diverse, that is they will be quite similar to their parents. While for two-point crossover, offsprings will be relatively more diverse compared to one-point crossover. So two-point crossover will be more likely to lose good genes from parents and thus produce worse offsprings. One obvious limitation in our work is that the initial population size and the generation number are both too small. Normally these two values are in the range of 20~100 and 100~500 [11]. But in this case the iteration times of GA will increase a lot because larger population size and generation number will create larger (hyper parameter) search space. And we suppose larger-scale GA should be run on Ubuntu, which would be more efficient because the related framework supports that system to use GPU, unlike the limitation with macOS system.

3.3 Causal index and knowledge distillation experiment

Below is the image of causal index of feature indexed with 0 for characteristic pattern 0, 1, 2, 3, 5, 6, 7, 8, 9, 10. Here type 4 is missing because the NN trained for calculating causal index of feature 0 can only indentify 10 types except for type 4.



Below is the image of causal index of feature indexed with 538 for characteristic pattern 0, 1, 2, 3, 5, 6, 7, 8, 9, 10. Here type 4 is missing because the NN trained for calculating causal index of feature 538 can only indentify 10 types except for type 4.



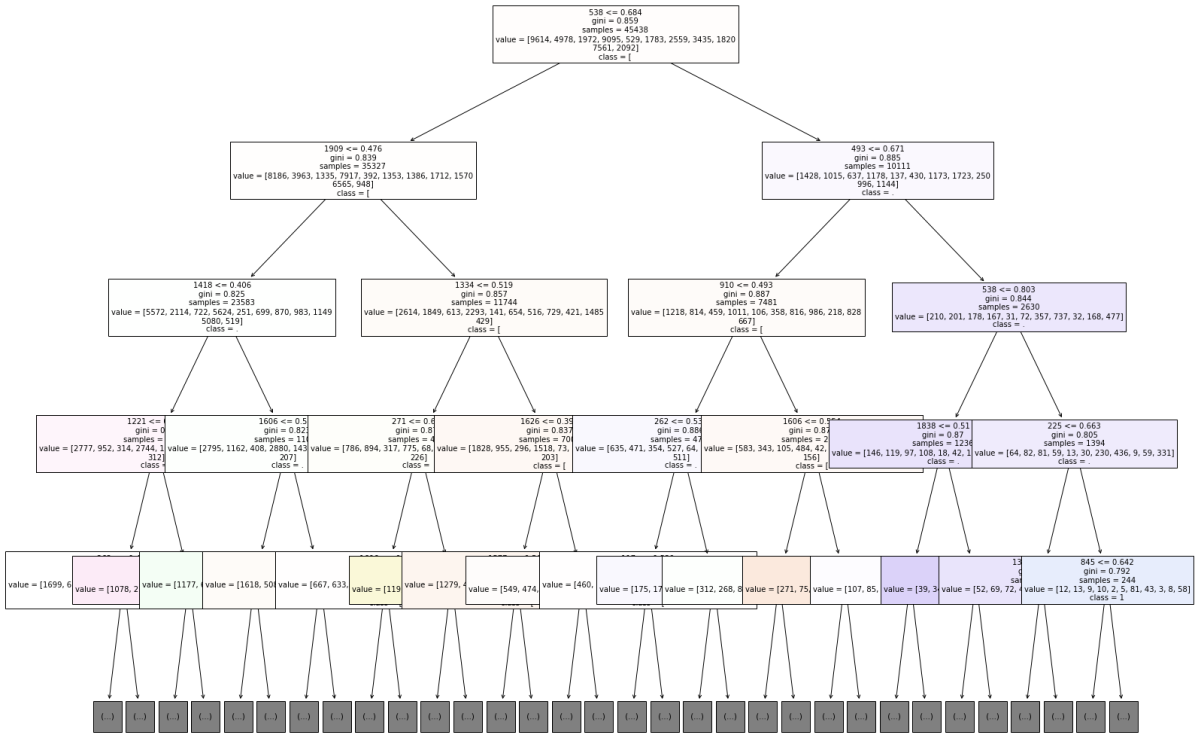
By observing the reverting points (where causal index reaches 0 and tends to be gentle after reaching a positive or negative peak that is relatively sharp) in the above 18 images, we noticed that there is not only one such reverting point in each images. So this means that both feature 0 and feature 538 appear in the extracted rules for this NN. Also as the two groups images of causal index are quite similar, they are both the plots with not quite smooth lines crossing zero (x-aixs) and some sharp peaks. So it seems feature 0 and feature 538 are equally important in the rule system. Also, we randomly selected other 10 features, then computed and plotted their causal index. We got the similar results as these two above images look like. So we inferred that each feature appears in the extracted rules and they are of equal importance in decision making of NN. However, we didn't compute and plot the causal index of all the 2048 features because that would take a quite long time to plot them all. So this conclusion is just a inference, and we used other methods to verify this inference, like the below decision tree implementation and randomly select one feature out of 2048 features, trained the model only on this feature and recorded its accuracy. For the latter method, we compared the accuracy of models taking different single feature as input, and found they are quite similar: they are all around 20% ~ 21%. And this finding confirmed our guess to some extent.

Also to check whether the extracted rules are good enough to describe the NN, we need to find all the features mentioned in the rules, and ignored those are not in the rules. However, it wasn't feasible to extract all the rules because we had to run the causal index experiment on 2048 features. And without running on GPU, that was actually too time-consuming. So instead of evaluating how good the extracted rules can explain the NN, we verified this inference: each feature are of similar importance for decision making. So we randomly picked 10 features, and trained the model on them one by one, we recorded the accuracy of each model. Finally, there was an interesting finding: the accuracy is all round 21% and we found out the specific class labels that each model can identify. We found that each of those models can identify one or two classes, and

Table.3

Model	accuracy
Original model trained with 2048 features	Around 45%
Model trained one feature	Around 21% (average accuracy of models with input of 10 randomly selected one feature
Model trained 100 features	Around 30% (average accuracy of models with input of 5 groups of randomly selected 100 features

After studying the causal index based on the characteristic input, we built decision tree models to verify our guess about the finding of the causal index study. We trained deicison tree models on 2048, 100, 50, 10 feature inputs respectively and the models' performance is shown in table2. Also, we visiualised the deicison tree with 10 feature inputs and only plotted the tree structure of 4-layered depth. The first line in the text box of each node is the extracted rule and the node color represents the class/category (label). For instance, in the parent node, the rule is that "X[538]<=0.684". We didn't plot a deeper tree structure because it was quite time-consuming as the number of nodes in each layer is at an exponential growth rate when we went deeper. And that's one of our limitations.



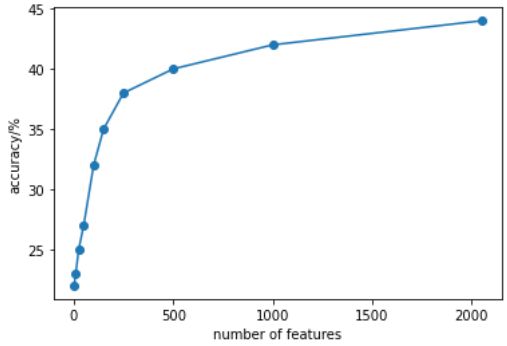
From the above visualised tree structure, we can see that there is barely no feature appears more than once in the rules attached to those visualised nodes. Only the feature indexed by 538 appears twice. But as we previously plotted the relationship between its causal index and the input, we knew there is not much difference between this feature and the others. Also, the distribution of the indiced of those features appearing in the visualised rules is relatively even in the range [0,2048]. This indicates that it is quite unlikely consecutive parts of the 2048 features or the concentration of the 2048 features will be prominent and outweigh the other features in the decision making process of NN. So the visualisation verifies our inference that almost every single feature appears in the rules and is weighted equally in the explanation mechanism of the NN.

As we can see from table 2, even we use all the given 2048 features, the accuracy of decision tree model is still not quite idea. And that accuracy is much worse than that of two layered NN. So though the decision tree has good interpretability, it doesn't have good accuracy on relatively complex classification problem. And the possible reason behind this is that a decision tree inherently discarded the input features that it doesn't find useful while NN will use them all unless we make some artificial change to the features like feature selection as pre-processing work. Because these two models work based on different principles: decision tree use inherently batch-learning algorithms and NN is based on the idea of gradient descent. So when being provided with enough training data, NN would be more accurate because it can model more arbitrary and complex functionsn(non-linear ones) unless overfitting happens.
In our experiment, we applied PCA to reduce the input dimension, in other words, use fewer features but more representative features as input. PCA, principal component analysis, the essence is dimensionality reducing and space transformation. Just like a data point was transformed from a 3D space to a 1d straight line, it has exactly the same principle. When transferring to another space, the features sharing similarity gather together and those with varying property will separate apart. So the motivation for PCA is: high dimensional data, such as images, is hard to analyse, interpretate, and visualize, and expensive to store, but luckily, high-dimensional data is frequently overcomplete that is many dimensions are redundant and can be explained by a combination of other dimensions so PCA can provide us an efficient way to derive a compressed representation system. So to better conclude the general and complete image features of the original training set, we hope to find the "direction" which contain the most variance of facial features. And so we used PCA technique to reduce the input feature number from 2048 to 100, and even smaller like 50 and 10.

Tabel .2 accuracy of decision tree models when the input feature number varies

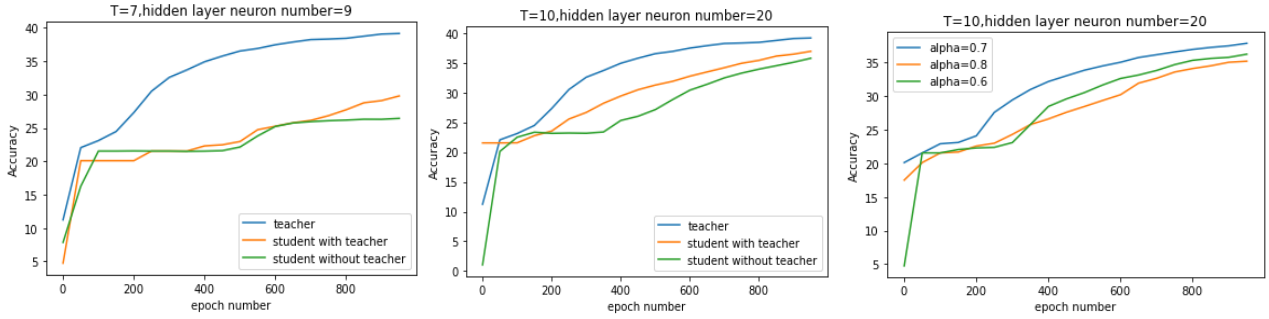
Input feature number	Decision tree model's accuracy (%)
2048 (all features)	0.1880200766081099
100 features (PCA)	0.12917712323339056
50 features (PCA)	0.12607317395324263
10 features (PCA)	0.12376172236164311

We drew the conclusion from the findings of causal index and the visualization of the decision tree: all the features are almost equally important as predictors. That is they all equally contributed to the decision making of NN. As there is not much difference between varying features and actually, this is reasonable because they are all synthetic, so it is not quite meaningful to discuss about the model's interpretability because there is no prominent feature for the NN's decision making as basically each feature appears in the extracted rules from the NN and the only difference is the threshold (reverting point) of related feature in each rule. Also, the features are synthetic so it is hard to interpret its actuall meaning. Instead, we may use this finding to compress the model. We explored how the model's performance changes with the change of input feature number. Below is the scatter plot of the relationship between the feature number and the model's accuracy:



We randomly choose 2048 (all features), 1000, 500, 250, 150, 100, 50, 25, 10 and 2 features and used them to train a two layered NN with the same acrchietecture and hyper parameters setting. And after 400 epochs, we obtained their corresponding accuracy, and drew the above scatter plot. In the scatter plot, we can see clearly that there is a critical point in the range of [250, 500]. This point is normally known as the "elbow" point as when the feature number is larger than that value, the accuracy doesn't decrease a lot as the feature number descreases. While when the feature number is smaller than that value, the accuracy drops down more quickly as the feature number gets smaller. This would be useful when making a balance between model accuracy and training time, we can decrease the feature number to 500 and at the same time ensure the accuracy doesn't drop too much.

So to largely compress the model while maintaining the accuracy as much as possible, we used the finding from the study of causal index, and the study of the impact of feature number on accuracy. We decided to use 500 features as input and for teacher net, we used a 3-layered teacher net with 500 neurons in hidden layer, 100 neurons in the first layer, 50 in second hidden layer, and 11 for output layer. And then for our student model, we developed two kinds of student models. They are all simpler two-layered model and the only difference is neuron number in hidden layer: 9 and 20. We made a comparison between a student net learnt from the teacher net and the student net without a teacher net. And to better illustrate how the accuracy has changed during the process of compressing the model, we also plot the techer's net accuracy. This is shown in the leftmost graph in the following figure. Besides, we explored the impact of the parmaters in Knowledge distillation. By comparing the left image with the middle one, we can see the performance of KD under different temperature with different hidden layer sizes. And in the rightmost image, we can observe that how the weight of soft loss function (L_{soft}) will influence the behaivour of KD.



So from the above 3 images, we can concluded that first, the student net learnt from teacher net outperform the same student net without leaerning from a teacher net. Secondly, when the size of student net grows, higher temperature will

be better, and we suppose that is because when the size increases, there are more useful information (dark knowledge) stored in the negative class labels, so we need a larger temperature to learn this. Bacially, choosing of temperature is to make a trade-off between these:

- If we need to learn more dark knowledge stored in the negative lables, then we should use higher T.
- If we hope to avoid some noises in the negative lables, then we should use lower T.

Then, for the impact of alpha, which is the weight of soft loss term in the distillation loss function, we found relatively smaller alpha tends to perform better, and that might because we need to make a balance between the knowledge and experience learnt from the teacher and the standard lables (standard answer). But generally, the weight of the previous one should be larger than that of the latter one because the knowledge and experience learnt from the teacher should be more prominent.

3.4 Results and Comparisons

To sum up, we will list the accuracy of the baseline model to compare with that of the model tuned by genetic algorithm and the model compressed by Knowledge distillation:

NN structure	Accuracy after 1000 epochs
Two-layered baseline NN	44.56%
Three-layered baseline NN	41.38%
Most optimal model tuned by GA	46.05 %
Most optimal compressed model	37.82 %

And the details of the most optimal model tuned by GA is

NN parameter	value
NN Structure	Two-layered
Input layer neurons	2048
Hidden layer neurons	310
Output layer neurons	11
Learning rate	0.0013280212483399733
optimiser	Adam

And the details of the most optimal compressed model:

Teacher:

NN parameter	value
NN Structure	Three-layered
Input layer neurons	500
Hidden layer neurons	100,50
Output layer neurons	11
Learning rate	0.001
Epoch number	1000
optimiser	Adam

student

NN parameter	value
NN Structure	Two-layered
Input layer neurons	500
Hidden layer neurons	20
Output layer neurons	11
Learning rate	0.001
optimiser	Adam
Temperature	10
Alpha in loss func	0.7

So generally, we can see for the baseline model, two layered one performs better and that may because of overfitting problem and may also because as the size of NN increases, so it needs longer training time to find the optimal.
The hidden layers are used for better classifying some boundary cases, and it is just like classifying a convex or concave region with arbitrary complexity in a plane. Increasing number of hidden layers will increase the robustness of the model. But, increasing number of hidden layers also means longer computation and training time and may even lead to overfitting

problem because in this case we have large amounts of training samples but only have 11classes which means if there is any bias on either of the classes, the difference would be amplified. That is the possible reason why the accuracy of these two models differs.
For the tuned model with genetic algorithm, we can see an improve in the accuracy. Though the accuracy doesn't enhance quite a lot, it still shows the power of GA. Because we also did lots of background study when constructing the baseline model and it is more difficult to enhance a good model. Also, due to the limitation of computation time and resources, we suppose if the generation number and the initial population size increase, then the optimal solution will be more ideal as the search space is enlarged. And we also learnt that the operators of selection, crossover and even mutation will influence the performance of GA so it is necessary to adjust these operators when implementing GA algorithm.
For the compressed model, we can see though there is a decrease in accuracy which is also inevitable, both the model structure and the size has been effectively re-scaled to a smaller and simpler one. We can see the hidden layer size shrinks more than ten times as much as the original one. So we believe this would help deploy vehicle classification models to edge devices like smart phones and will save large amounts of computation resources and time. Also,we learn that the temperature in loss function and archietecture of the teacher model will influence the effect of knowledge distillation, and we need to make a proper balance between the knowledge from the teacher and the ground truth labels. Besides, the power of KD will be more obvious in the compressing of a more comple NN like the CNN and etc.

4 Conclusion and Future Work

To sum up, the classification task of is perfectly done, since the best accuracy can reach more than 45%, which is larger than a random classifier. By applying genetic algorithm for finding the optimal solution of hyper parameters tuning, we successfully enhanced the accuracy of the baseline model. And also explored the impact of different operators in genetic algorithm. Then, by studying the characteristic input and causal index, we draw the inference that all of the 2048 fearures appear in the extracted rules and they are of equal significance for the decision making of NN. To verify that inference, we built decision tree model and the result confirmed our inference. And we also noticed that NN trained on one single input feature can identify one or two classes normally. Besides, we used PCA to reduce the input dimension, and studied the impact of input feature number on the classification accuracy. We observed that 500 input features is a good choice because it largely decreases the input size and at the same the accuracy doesn't drop too much. That can help to keep a good balance between computation resources and model performance. Finally, based on the previous study, we compress the model by reducing the structure's complexity and the size of hidden layer, and we efficiently compressed the original complex model to a smaller and simpler one. This makes the deployment of vehicle classification model to an edge device possible.

In future, as discussed previously, due to the limitation of computation resources and time, the population size and generation number should be larger. So in future, we will run the experiment on other system which better support framework on GPU rather than MacOS. And we believe this will largely increase the effect of GA for hyper parameter tuning. Also, in the study of causal index, we can only draw an inference rather than a strict and accurate conclusion because it is not that feasible to explore the causal index of all the 2048 features. So we will explore more on this topic: how we can better and more efficiently interpret a NN for a complex classification task. And so far, we have learnt that there are some methods to cope with this problem, and researchers bring up some thoughts to create an explanation mechanism for a neural network. One of the method is to derive an algorithm for extraction of decision trees from article neural networks, named as ANN-DT[13], which mainly implemented the combination of feedforward neural network and CART. Also, there are other methods like developing a NN and use decision tree to approximate the output of each neuron, one implemented related algorithm is HERETIC, using a symbolic learning algorithm (decision tree) on each unit of NN[14] and another method of increasing the interpretability of NN by pruning the NN to smaller size which will transform the NN to a extractable mode but will also reduce its prediction ability and robustness.
Also, for knowledge distillation, apart from using KD to compress the model, we will also explore how to use Soft Binary Decision Tree or NBDT to distill a decision tree out of NN. This will combines the advantage of a decision tree (high interpretability and fast convergence speed) and that of NN (high accuracy and robustness for coping with more complex classificaito problem).

References

1. Z. Luo et al., "MIO-TCD: A New Benchmark Dataset for Vehicle Classification and Localization," in IEEE Transactions on Image Processing, vol. 27, no. 10, pp. 5129-5141, Oct. 2018, doi: 10.1109/TIP.2018.2848705.
2. Yao, Y., Zheng, L., Yang, X., Naphade, M., and Gedeon, T., “Simulating Content Consistent Vehicle Datasets with Attribute Descent”, arXiv e-prints, 2019.
3. Khan, S.D. and Ullah, H. (2019). A survey of advances in vision-based vehicle re-identification. Computer Vision and Image Understanding, 182, pp.50–63.
4. deap.readthedocs.io. (n.d.). Evolutionary Tools — DEAP 1.3.1 documentation. [online] Available at: https://deap.readthedocs.io/en/master/api/tools.html#deap.tools.History [Accessed 2 Jun. 2021].
5. T. D. Gedeon, 2021 s1, NN3_mlpbp+relu_v8, lecture notes, Neural Networks, Deep Learning and Bio-inspired Computing COMP4660, The Australian National University, delivered February 26th, 2021.
6. T. D. Gedeon and H. S. Turner, "Explaining student grades predicted by a neural network," Proceedings of 1993 International Conference on Neural Networks (IJCNN-93-Nagoya, Japan), 1993, pp. 609-612 vol.1, doi: 10.1109/IJCNN.1993.713989.
7. Turner, H and Gedeon, TD.: Extracting Meaning from Neural Networks. Proceedings 13th Int. Con. on AI, Avignon, 1993.
8. Zhu, Q. (2020) "Predicted the authenticity of anger through LSTMs and three-layer neural network and explain result by causal index and characteristic input pattern," 3rd ANU Bio-inspired Computing conference (ABCs 2020), paper 83, 7 pages, Canberra.
9. Wang, L. and Yoon, K.-J. (2021). Knowledge Distillation and Student-Teacher Learning for Visual Intelligence: A Review and New Outlooks. IEEE Transactions on Pattern Analysis and Machine Intelligence, pp.1–1.
10. S. Bock and M. Weiß, "A Proof of Local Convergence for the Adam Optimizer," 2019 International Joint Conference on Neural Networks (IJCNN), 2019, pp. 1-8, doi: 10.1109/IJCNN.2019.8852239.
11. blog.csdn.net. (n.d.). [online] Available at: https://blog.csdn.net/carlyll/article/details/105900317?ops_request_misc=%257B%2522request%255Fid%2522%253A%252216227613741678026407286 6%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=162276137416780264072866&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduend~default-4-105900317.first_rank_v2_pc_rank_v29&utm_term=%E9%81%97%E4%BC%A0%E7%AE%97%E6%B3%95%E8%B0%83%E5%8F%82+%E5%8F%82%E6%95%B0%E8%AE%BE%E7%BD%AE&spm=1018.2226.3001.4187 [Accessed 2 Jun. 2021].
12. Jinghui Zhong, Xiaomin Hu, Jun Zhang and Min Gu, "Comparison of Performance between Different Selection Strategies on Simple Genetic Algorithms," International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06), 2005, pp. 1115-1121, doi: 10.1109/CIMCA.2005.1631619.
13. G. P. J. Schmitz, C. Aldrich and F. S. Gouws, "ANN-DT: an algorithm for extraction of decision trees from artificial neural networks," in IEEE Transactions on Neural Networks, vol. 10, no. 6, pp. 1392-1401, Nov. 1999, doi: 10.1109/72.809084.
14. M. R. A. Iqbal, "Eclectic rule extraction from Neural Networks using aggregated Decision Trees," 2012 7th International Conference on Electrical and Computer Engineering, 2012, pp. 129-132, doi: 10.1109/ICECE.2012.6471502.